A new paradigm is among us. Along with the cloud and mobile devices, nonrelational, distributed database management systems (non-RDBMS) are now growing in popularity. Examples of these database implementations include NoSQL (not only SQL) offerings such as Cassandra, CouchDB, FlockDB, GraphDB, Hibari, MongoDB and SimpleDB. Although non-RDBMS have been around for the better part of a decade, their popularity has grown to the point at which information systems professionals should become familiar with their architecture and use, how to assess their risk posture, and how to secure them. This article provides an overview of this technology, an idea of the security vulnerabilities inherently found in these technologies and guidance on how to remediate those vulnerabilities.

**NON-RDBMS VALUE-ADD**

Once thought of as a technology solely for academia, non-RDBMS are now reaching critical mass in industry. Leading technology service providers (e.g., Twitter) have begun to use them, and individuals and companies consume those provider offerings. Non-RDBMS are becoming the preferred database architecture for organizations using Web 2.0 technologies due to the open-source nature of these platforms, which leads to cost savings because organizations do not have to invest in traditional relational database software licensing and/or local hardware. Concurrently, large enterprises are using non-RDBMS to augment their existing RDBMS investments for storing and analyzing big data. Big data, as defined through Oracle, includes the aggregation of an organization’s traditional, sensory (e.g., log data, metadata) and social (media) data. Beyond cost savings, organizations can expect to experience enhanced scalability, elasticity (called sharding in the non-RDBMS world), modularity, portability and interoperability while using non-RDBMS platforms with Web 2.0 technologies, such as Ruby on Rails (a Web 2.0 programming language focused on dynamic content) or web services/service-oriented architecture (SOA).

As shown in figure 1, NoSQL solutions, such as Couchbase, allow for enhanced system scalability and elasticity (i.e., sharding).

It is important to understand how non-RDBMS differ from relational database management systems (RDBMS), as this new technology platform provides a different set of strengths, weaknesses, opportunities and threats (SWOT) for the enterprise. Non-RDBMS provide enhanced elasticity, modularity, portability and interoperability as strengths and opportunities. While it lacks basic access controls, authentication, auditing (logging) or configuration management are the weaknesses and threats with non-RDBMS. The differences between non-RDBMS and RDBMS include:

- Data in non-RDBMS are distributed across multiple computers, which could upset the organization’s state of privacy compliance as the data may be spanned/sharded across multiple jurisdictions.
- Data are created, read, updated and/or deleted (CRUD) from/to non-RDBMS through an application programming interface (API) call vs. a database connection (e.g., Open Database Connectivity [ODBC], Java Database Connectivity [JDBC]).
- Non-RDBMS differ from RDBMS in the way that they treat data; for example, tables in a distributed database system (DDS) are referred to as domains/namespace.
- In non-RDBMS, Data Definition Language (DDL), or metadata, is not as easily queried as in RDBMS.
- Most non-RDBMS have moved away from using Structured Query Language (SQL) for Data Manipulation Language (DML) calls; many now use a DML platform called NoSQL.
- Non-RDBMS require an operating API service to be running as opposed to a database server instance, which often leads to a considerably lower operating expense (OpEx).
By running an API service, such as Cloudian’s, non-RDBMS users can save on capital expenses (CapEx) and operating expenses (OpEx), and they can easily build applications using the non-RDBMS technology. Cloud service providers (CSP), such as Amazon, are counting on this. Amazon’s Web Services’ (AWS) portfolio includes an offering called SimpleDB, which is used by various start-ups (e.g., Flipboard, Kehalim, Livemocha and LOUD3R) for an extremely fast time to market.4 Non-RDBMS consumers connect their NoSQL databases to web applications by using third-party products such as Cloudian’s (see figure 2) or writing their own.

**NON-RDBMS TYPES**

Non-RDBMS, specifically NoSQL implementations, can be broken down into four categories: column-oriented/tabular (e.g., Cassandra), document-oriented (e.g., CouchDB, MongoDB, SimpleDB), graph databases (e.g., FlockDB, GraphDB) and key-value databases (e.g., DynamoDB, Hibari).

Column-oriented/tabular flavors use database table-like structures without the joins often found in a relational implementation, while document-oriented flavors store web-centric documents in the format of either JavaScript open notation (JSON) or XML-based documents. The closest relative to the relational database for NoSQL implementations is the key-value flavor, which uses primary keys such as in an RDBMS but without a table.5

Finally, there is the graph database, which is based on the Graph theory. This theory states that relations exist between attributes and entities.6 For example, a married couple is related by their marital status, which is an attribute/adjunct vs. an entity/table.
NON-RDBMS CHALLENGES

Although non-RDBMS are becoming increasingly popular, they have their challenges. The largest challenge found in non-RDBMS is their distributed nature, which in today's highly regulated and litigious society can be a burden too high for many organizations to bear. Data that are distributed across multiple locations are a problem in themselves, and this issue can be exacerbated if the location of a specific record cannot be determined when it is needed for audit, e-discovery or forensic purposes. Several other vulnerabilities can be found within non-RDBMS, including:

- Lack of uniform standards across non-RDBMS, affecting portability
- Lack of supported authentication methods
- Limited consumer-level logging/monitoring support
- Limited support for dynamic application security testing (DAST) by vulnerability assessment tools
- Incorrect belief that NoSQL-based non-RDBMS implementations are more secure than RDBMS

Non-RDBMS using NoSQL are still susceptible to input validation vulnerabilities such as SQL injection. Therefore, it is important that the source-code check for input validation occurs at both the client and server side, because most DAST will not pick up the nuances of SQL injection-based vulnerabilities for non-RDBMS. Organizations that are obligated to comply with, for example, the Payment Card Industry Data Security Standards (PCI DSS) and the US Health Insurance Portability and Accountability Act (HIPAA) should also be cognizant of the fact that non-RDBMS provide limited logging for auditing and accounting purposes because the logs are built predominantly for debugging. Authentication for non-RDBMS is also limited; therefore, the onus is on the programmer to ensure that users are authenticated on the front end of the system (e.g., www.mysite/login.php). Finally, as there is no standard for non-RDBMS, organizations should know that there will be limited portability of the implementation between non-RDBMS implementations/providers.

Beyond the known portability limitations, additional vulnerabilities can be found from a process standpoint. These vulnerabilities can be caused by a lack of documented and/or implemented security review tasks within the software development life cycle (SDLC), or they can be due to a lack of a separation/segregation of duties. A large part of this problem stems from the ability of a developer to sidestep the traditional role and/or need of a database administrator (DBA) to design and build a logical/physical RDBMS. The cause of this is the reliance on the API calls vs. a prebuilt database to connect to via ODBC/IDBC. Additionally, because small start-ups are the most prevalent users of this technology, there are often control gaps from a security standpoint. With their often limited budgets and staff, these smaller entities often lack a documented and secure SDLC, which leads to increased risk. However, it has been found that, with some education and awareness, these vulnerabilities can be remediated relatively quickly. Remediation can be done in a timely and cost-effective manner through process and technology-based controls/safeguards. Examples include a proper segregation of duties for development and production environments; a formal SDLC process that incorporates security thought leadership, such as Microsoft's Security Development Lifecycle (SDL); or implementing various firewall implementations (e.g., web application firewalls [WAF], eXtensible Markup Language [XML] firewalls and database firewalls).

Whether an organization adapts these processes to remediate known vulnerabilities, or whether it implements technological solutions, there are ways to mitigate the risk.
found in implementing non-RDBMS. The largest remediation strategy should be to do a risk assessment before using non-RDBMS. Once the risk assessment is completed, a go or no-go decision can be made to determine the risk appetite for non-RDBMS. This assessment should also determine which data, if any, may be stored in non-RDBMS. For many organizations, the data stored in non-RDBMS are sensory (log) data from multiple sources. The non-RDBMS data store is then used by security information and event management (SIEM) software packages for analytical purposes (e.g., trend analysis for detecting security incidents). Technical controls/safeguards can also be used to remediate the risk introduced when using non-RDBMS. Specifically, an organization may implement a WAF or an XML firewall as a preventive control while using non-RDBMS. In the end, an organization needs to do a cost-benefit analysis (CBA) on using non-RDBMS while incorporating the necessary controls to safeguard data.

CONCLUSION
By completing a CBA, conducting a risk assessment and understanding the vulnerabilities found within this platform, an organization can make an informed decision about whether or not to use non-RDBMS. This decision is more important now than ever, as organizations must realize that data are their lifeblood, and as data grow, the manner in which they are treated should change. Hence, non-RDBMS now exist to either support Web 2.0 applications or augment RDBMS for managing big data. As organizations collect data in staggering quantities, it is crucial that they keep their options open for the best system to use for storing these data. As non-RDBMS grow toward critical mass, it is important to know how they work, how they can be used within an organization, how they differ from RDBMS platforms, and who is best suited to use them for collecting and processing data.

ENDNOTES
6 Finley, Klint; 5 Graph Databases to Consider, 20 April 2011, www.readwriteweb.com/cloud/2011/04/5-graph-databases-to-consider.php