Detecting Cypher Injection with Open-Source Network Intrusion Detection

Author: Michael Dunkin

Abstract

Security researcher John Lambert once said, “Defenders think in lists. Attackers think in graphs” (Lambert, 2015), but attackers do not simply *think* in graphs; they can *attack* graphs using a technique called Cypher injection. Cypher, a language used to query graph databases such as Neo4j, is vulnerable to a class of attacks called query injection. Cypher query injection allows attackers to gain unauthorized access to graph data by injecting unexpected instructions into query input. Graph databases represent tempting targets because they support critical applications such as fraud detection, medical contact tracing, and the well-known security product Bloodhound. Yet, they are not as welldefended as traditional relational databases. Researchers published over 60 papers in 2023 alone about SQL Injection, the better-known relative of Cypher Injection that targets relational databases. However, the first academic evaluation of Cypher injection appeared only in November 2023, and as of March 2024, there was no publicly available detection for Cypher injection. This study describes a set of rules written for open-source network intrusion detection systems that can detect Cypher injection with an accuracy of over 90%.

**1.Introduction**

In 1945, John Von Neumann had an insight foundational to general-purpose computing: both computer instructions and the data they act on can reside in the same store (von Neumann, 1945). Put another way, code and data may behave differently, but they look alike. As Scott Shapiro (2023) observed, "Code instructs, data represents. If you want to take some action based on conditions, use code. If you want to represent the state of the world, use data. Mix the two up and you’re in trouble.” Starting as early as the 1988 Morris Worm, attackers have been using Von Neumann’s insight against their victims through a technique called code injection (Shapiro, 2023). Code injection represents a class of attacks that involve submitting code into an application’s input in a manner that forces the receiving application to run that code. A variant of code injection called SQL query injection emerged after 1998 to become among the leading types of application vulnerability (Rawat et al., 2023). Over the past two decades, researchers have created a robust set of detections for SQL injection, ranging from signature-based detections to machine-learning models. While SQL injection targets traditional relational databases, the rise of alternative NoSQL databases in the past decade has led to a corresponding rise in query injection attacks that target those databases. In 2021, practitioners first described a new variant of query injection called Cypher injection that uses the Cypher query language to attack graph databases such as Neo4j. As of March 2024, no detections are publicly available for Cypher injection attacks.

This paper proposes a set of open-source network intrusion detection rules to identify Cypher injection requests. It then describes a method for testing the efficacy of Cypher injection detections using a battery of test cases that represent each class of query injection along with legitimate Cypher queries drawn from a sample financial application and the official Cypher language guide. The rules proposed in this study detect Cypher injection with an accuracy of over 90%

**.1. Prior research on query injection attacks**

**1.1.1. SQL query injection**

In 1998, a researcher discovered a variant of code injection called SQL injection that targets relational database systems by appending code written in the SQL query language to data input (Forristal, 1998). Attackers have exploited SQL injection vulnerabilities in high-profile breaches of organizations such as TJX (Buley, 2009), Sony Pictures (Federal Bureau of Investigation, 2011), and LinkedIn (SentinelOne, 2022). SQL injection has remained among the top application security vulnerabilities (Rawat et al., 2023), leading to hundreds of academic works devoted to detecting and preventing these attacks.

**1.1.2. NoSQL query injection**

In the late 2000s, relational databases began to show limitations in storing big data, providing consistency models that supported large, distributed platforms such as social media, and representing social networks. Non-relational NoSQL databases such as document, columnar, key-value, and graph databases aimed to fill the gaps left by traditional table-and-column relational databases. SQL is a query language used to access data from traditional relational databases. One common trait of these new database classes was that, at the time, they did not support SQL, so they were, by definition, immune from SQL injection attacks. These non-relational databases were referred to collectively as NoSQL (Not Only SQL) databases, reinforcing the misconception that they were safe from injection attacks. However, by 2015, researchers discovered that the most popular class of NoSQL databases, document databases such as MongoDB, is vulnerable to JSON query injection (Ron, 2015).

**1.1.3. Cypher Query Language Injection**

Researchers have made strides over the past few years in detecting and preventing injection attacks against document databases (Hou et al., 2016), (Boza & Muñoz, 2017). However, researchers have only recently begun to focus on code injection attacks that exploit vulnerabilities in the Cypher query language used to access graph databases such as Neo4j, AWS Neptune, and Redis Enterprise Graph.

In 2021, practitioners began exploring Cypher injection in blog posts (Chivato, 2021). The vendor Neo4j described protections against Cypher injection that year in its knowledge base (Bowman & Lamont, 2021). By that summer, a Burp Suite contributor had created a Burp Suite extension that can scan an application to determine whether it is vulnerable to Cypher injection (Morkin1792, 2021). A 2022 blog post by M. Elgharbi, based on a presentation at the 2022 BSides Orlando conference (Elgharbi, 2022a), enumerated examples of Cypher injection that form the basis for many of the specific Cypher injection techniques used in this paper (Elgharbi, 2022b). A 2023 blog post by a security practitioner also provides a helpful list of Cypher detection examples that inform the samples in this study (Bachrach, 2023). The first academic exploration of Cypher injection (Van Landuyt et al., 2024) recently demonstrated several variants of Cypher injection.

**1.2. Graph Databases and the Cypher Query Language**

**1.2.1. Introduction to graph databases**

Relational databases, which organize data into traditional tables with rows and columns, are based on 19th-century set theory. They describe relationships between objects in terms of membership in collections of sets. Graph theory, developed in the 18th century, provides an alternative approach to describing relationships between objects. In attempting to solve The Seven Bridges of Konigsberg problem, Leonard Euler developed a theory that uses graphs to represent objects, known as vertices or nodes, and relationships between them, known as edges (Paoletti, 2011). These graphs and the mathematics that developed around them are uniquely suited to applications such as finding the shortest path between two points on a map or finding the most connected person in a social network. Researchers were laying the groundwork for graph databases by the 1970s (Phillips, 1977), but only with the advent of the NoSQL movement did graph databases become commercially viable. In 2011, Neo4j, maker of the leading graph database of the same name, developed a query language called Cypher that allows users to represent graph queries as ASCII art. In 2015, Cypher was adopted as an open standard

and is now used to query several graph databases, including AWS Neptune, Redis Enterprise Graph, and Apache Spark.

**1.2.2. Applications of graph databases**

Graph databases represent tempting targets for attackers. Cypher-based graph databases are used in security solutions from Bloodhound (Lemmens, 2021) to MITRE CyGraph (Noel et al., 2016). Financial organizations use Cypher to detect fraud and money laundering (Chung, 2022), while healthcare organizations use it to track clinical trials (Murali et al., 2022) and perform contact tracing (Mao et al., 2021). By detecting and preventing Cypher injection, organizations can protect the graph databases' confidentiality, integrity, and availability. More importantly, by detecting even unsuccessful Cypher injection attacks, organizations become aware of the presence of attackers in their networks.

**2. Research Method**

**2.1. Mechanics of a Cypher Injection Attack**

Applications that use input to construct Cypher queries and do not heed Neo4j’s recommendation to parameterize and cleanse input (Bowman & Lamont, 2021) are vulnerable to Cypher injection. The simplest form of Cypher injection is the tautology. Like a tautological SQL injection, a tautological Cypher injection appends a statement that is always true, such as “1=1”. The attack appends the tautology by first closing a quoted string.

**2.2. Types of Cypher Injection Attacks**

In 2006, researchers classified in-band SQL injection attacks into the following seven categories (Halfond et al., 2006):

● Tautologies

● Illegal/Logically Incorrect Queries

● Union Query

● Piggy-backed Queries

● Stored Procedures

● Inference

● Alternate Encodings

Researchers further classified these injection attacks to include the following categories (Clarke et al., 2009):

1. In-band

a. Error-based

b. Union-based

2. Inferential (blind)

a. Boolean-based

b. Time-based

3. Out-of-band (blind)

While Cypher’s syntactical ordering and inclusion of ASCII art set it apart from SQL, it does borrow some elements from SQL to speed its adoption by developers familiar with relational databases. Because of these similarities, many of the categories of SQL injection have their analogs in Cypher injection. The first academic treatment of Cypher injection (Van Landuyt et al., 2024) adopted the classification of in-band injection attacks described by Halfond et al. in 2006. Elgharbi explained in 2022 that most of the broader classes of SQL query injection described by Clarke apply to Cypher injection (Elgharbi, 2022b). Specifically, the only class of query injection attack that does not apply to native Cypher is the time-based injection.

Marcussen (n.d.) and Bowman & Lamont (2021) demonstrated examples of Cypher injections that do not fit neatly into any of the categories above. This new category of injections focuses on evading detection. Combining the taxonomies of Halfond et al. (2006) and Clarke et al. (2009) and adding detection evasion yields the following taxonomy of Cypher injection techniques. **2.3. Query Injection Detection**

Researchers had already begun building detections for SQL injection within four years of its discovery (Lee, S.Y. et al., 2002). The pace of those investigations has only intensified in recent years as SQL injection detection has become a popular application of machine learning for security researchers. In 2023 alone, researchers published over 60 academic articles on SQL injection detection, at least 35 involving machine learning models. As Sadeghian et al. (2013) demonstrate in their summary of these new detection methods, older signature-based detections are easier to evade than detections based on machine learning models. Despite those limitations, the intrusion detection rules proposed in this paper will implement signature-based detections to provide an adequate initial set of detections that organizations can implement in open-source intrusion detection systems. Future researchers may build on these detections and the sample Cypher queries described in this study to create more robust detections using machine learning models.

**2.4. Queries to Test Cypher Injection Detection Efficacy**

**2.4.1. 84 Cypher injection test cases**

To determine the true positive rate of proposed open-source intrusion detection rules, this study relies on 84 published examples of Cypher injection from seven sources. Some of these examples required interpretation to convert into syntactically correct Cypher commands. Note that some of these samples destroy data, so exercise caution when running them against live graphs.

**2.4.2. 155 benign Cypher query test cases**

To determine the false positive rate of proposed open-source intrusion detection rules, this research gathered 155 valid test cases that cover all Neo4j syntax along with specific examples from a sample Neo4j fraud detection application. To ensure completeness, many of these test cases derive from Neo4j’s language guide cheat sheet (Neo4j, Inc., n.d.), which lists every clause in the Neo4j language. To add complexity and verisimilitude to the test set, the author added Cypher queries typically used by a published sample fraud detection application (Bastani, 2013).

**2.5. Test Environment**

These tests ran in a virtualized lab environment consisting of a client, a firewall, and a data server. Each of the three machines in this environment ran on RedHat Enterprise Linux 9.3 inside a VMWare virtual machine controlled by VMware Workstation Player 16.2.4. Details of the processes and data structures used in this test environment appear in a GitHub repository

**2.5.1. The client sent Neo4j queries and recorded observations**

A bash process running on a client virtual machine retrieved test Cypher queries stored in a MySQL database, then used Neo4j CypherShell to transmit those queries to the Neo4j graph database. It then inspected a Snort intrusion detection system log to determine whether a Snort rule had detected Cypher injection. Finally, the process recorded the results of the observation in a database of observations running within MySQL. Both MySQL databases resided in an instance of MySQL 8.0.2 running on the client virtual machine.

**2.5.2. The data server ran a fraud detection Neo4j graph database**

An instance of Neo4j 5.16.0 Community Edition ran on a virtual machine within a dedicated virtual network. This Neo4j instance implemented a sample financial fraud detection graph provided by Bastani (2013). See Bastani (2013) for an explanation of this fraud detection graph.

**2.5.3. PfSense firewall limited access to the data server network**

A PfSense 2.7.2 firewall running on a virtual machine limited access to the data server’s dedicated virtual network. PfSense is a leading open-source firewall that can run in a virtual environment. PfSense supports Snort, a leading open-source Network Intrusion Detection and Prevention System. Snort provides a comprehensive set of standard intrusion detection rules. While a paid version of Snort provides access to the most up-to-date rules, the free version used in this study has access to rules that may lag by as much as 30 days. The rule set in this study was current as of January 12, 2024, at 10:43 pm GMT. This instance implemented every rule available in the following Snort rule sets:

● Snort subscriber rules

● Snort GPLv2 Community Rules

● Emerging Threat Open Rules

● Sourcefire OpenAppId detectors

● FEODO Tracker Botnet C2 IP Rules

**2.6. Control and Test Groups**

As a baseline, this experiment ran all test cases through the test process with all the available snort rules described above enabled but with no additional custom Snort rules. In the control run, no existing Snort rules as of January 12, 2024, flagged any of the 239 test cases as potentially malicious. The average duration for a Cypher query in the control run was 4,183 milliseconds. The process and infrastructure used to test opensource Cypher injection detections were identical to those used in the baseline, with one key difference— additional custom Snort rules.

**2.7. Test Procedure**

**2.7.1. Stage a proposed Snort intrusion detection rule**

After creating a Snort rule to detect Cypher injection, the tester applied the rule to the pfSense Snort service’s “custom.rules” category on the interface that limits access to the data server. To ensure consistent results, the tester restarted the pfSense firewall and confirmed that the Snort service was active on all network interfaces. Section 3.1 below describes the proposed detection rules.

**2.7.2. Run test process**

A Bash script tested the efficacy of proposed open-source Cypher injection detections. First, it retrieved each of the 239 test Cypher queries stored in the request MySQL relational database table. For each Cypher query, the test process used CypherShell to send the query to the fraud detection Neo4j graph database. It then recorded the database response and the query duration. Next, the test process waited 60 seconds to ensure that the pfSense Snort service had written its detection to a detection log file, then read the log file to determine whether the proposed detection flagged the query as Cypher injection. Finally, the process recorded the results in a table of observations stored in MySQL