CyGraph: Graph-Based Analytics and Visualization for Cybersecurity

S. Noel[1], E. Harley, K.H. Tam, M. Limiero and M. Share
The MITRE Corporation, McLean, VA, United States
[1]*Corresponding author: e-mail: snoel@mitre.org*

ABSTRACT

This chapter describes *CyGraph*, a system for improving network security posture, maintaining situational awareness in the face of cyberattacks, and focusing on protection of mission-critical assets. CyGraph adopts a unified graph-based cybersecurity model relevant to potential and actual cyberattacks, defenses, and mission impacts. It captures incremental attack vulnerability, security events, and mission dependencies within a network environment, builds a predictive model of possible attack paths and critical vulnerabilities, and correlates events to known vulnerability paths. It also includes dependencies among mission requirements and network assets, for analysis in the context of mission assurance. The resulting knowledge graph captures the complex relationships among entities in the cybersecurity domain. CyGraph brings together isolated data and events into an overall picture for decision support and situational awareness. It prioritizes exposed vulnerabilities, mapped to potential threats, in the context of mission-critical assets. In the face of actual attacks, it correlates intrusion alerts to known vulnerability paths and suggests best courses of action for responding to attacks. For postattack forensics, it shows vulnerable paths that may warrant deeper inspection. CyGraph also supports CyQL (CyGraph Query Language), a domain-specific query language for expressing graph patterns of interest, with interactive visualization of query results. To help manage visual complexity, CyGraph supports the separation of graph models into interdependent layers. For time-dependent graph models, it provides dynamic visualization of evolving graph state. CyGraph also integrates with third-party tools for visualizing graph state changes (e.g., driven by simulations). Furthermore, it has capabilities for synthesizing graph models with particular statistical properties.

Keywords: Cybersecurity modeling, Situational awareness, Mission assurance attack graphs, NoSQL graph databases, Graph visualization

Handbook of Statistics. http://dx.doi.org/10.1016/bs.host.2016.07.001

1 INTRODUCTION

Cyberattacks and defenses against them are conducted in complex environments, with numerous factors contributing to attack success and mission impacts. Network topology, host configurations, vulnerabilities, firewall settings, intrusion detection systems, mission dependencies, and many other elements can play parts. To go beyond rudimentary assessments of security posture, organizations need to merge isolated data into higher-level knowledge of network-wide attack vulnerability and mission readiness in the face of cyber threats.

Network environments are always changing, with machines added and removed, patches applied, applications installed, firewall rules changed, etc., all with potential impact on security posture. Intrusion alerts and antivirus warnings need attention, and even seemingly benign events such as logins, service connections, and file share accesses may be associated with adversary activity.

The problem is often not lack of available information, but rather the ability to assemble disparate pieces of information into an overall picture for situational awareness, optimal courses of action, and maintaining mission readiness. Security analysts and operators can be overwhelmed by a variety of consoles from multiple tools; each tool provides only a limited view of one aspect of the overall space under consideration. Tools such as security information and event management (SIEM) can help by normalizing data and bringing it together under a common framework. But the data still remain as individual pieces of information, rather than a comprehensive model of network-wide vulnerability paths, adversary activities, and potential mission impacts.

Our goal is to maximize the ability to discover potential threats and mission impacts, while minimizing the time needed for organizing multiple disparate data sources into meaningful relationships. For example, in the well-publicized Target retailer data breach (Harris and Perlroth, 2014), it was revealed that cyber defenders were actually aware of an alert for a particular aspect of the attack, but decided that it was a false positive. We could surmise that if those defenders understood the potential downstream ramifications of that alert, they would have considered it much more carefully, preformed additional investigations, etc. The goal is to provide the higher-order correlations that defenders need for truly informed decisions.

For the Target data breach, the attack began with a compromise within a partner (contractor) network. A common way for this to happen is through Trojan malware. Alerts for such malware are happening with high frequency in many environments and are often considered a low business risk (i.e., mainly a risk for individual clients). However, in the case of the Target breach, the infected host in the contractor became a launching point into the Target network. Several other steps were part of the breach, in which the attackers incrementally increased their scope of control, until they met their attack goals (exfiltrating large-scale credit card data).

The key lesson is that there were multiple attack steps, with multiple corresponding opportunities for detection. However, such alerts and other indicators occur within a large background of event noise. Since it is not practical for human defenders to consider all the possible multistep inferences, this needs to be automated. Also, defenders can make even more informed decisions (and reduce numbers of truly critical incidents to consider) by focusing such inference on mission-critical network assets. This can also be done preemptively, to discover and reduce such critical vulnerability paths.

To help address these challenges, we introduce *CyGraph*, a tool for cyber warfare analytics, visualization, and knowledge management. CyGraph brings together isolated data and events into an ongoing overall picture for decision support and situational awareness. It prioritizes exposed vulnerabilities, mapped to potential threats, in the context of mission-critical assets. In the face of actual attacks, it correlates intrusion alerts to known vulnerability paths and suggests best courses of action for responding to attacks. For post-attack forensics, it shows vulnerable paths that may warrant deeper inspection.

CyGraph builds an attack graph model that maps the potential attack paths through a network. This includes any network attributes that potentially contribute to attack success, such as network topology, firewall rules, host configurations, and vulnerabilities. The dynamically evolving attack graph provides the context for reacting appropriately to attacks and protecting mission-critical assets. CyGraph then ingests network events such as intrusion detection alerts and other sensor outputs, including packet capture. It also incorporates mission dependencies, showing how mission objectives, tasks, and information depend on cyber assets.

CyGraph fuses information from a variety of data sources to build its unified graph-based model. As shown in Fig. 1, this is a layered model, which includes the comprehensive information needed for making informed judgments about mission readiness in the face of cyber warfare.

The *network infrastructure layer* captures how the network is segmented and organized topologically, the locations of sensors, etc. The *cyber posture layer* considers elements within the network infrastructure that might impact cyberattacks/defenses, e.g., host configurations, vulnerabilities, services, shared resources, firewall policies, etc. The *cyber threats layer* describes potential adversary threats, for application against the defensive posture. This includes threat intelligence (e.g., shared among trusted partners) as well as event streams of alerts and other behavioral indicators. Finally, the *mission dependencies layer* captures dependencies among various mission components (from high-level objectives to tasks that support objectives to information required for task, etc.), as well as the particular cyber assets that support the mission components.

CyGraph has the potential for dramatically shortening the analytical cycle. It provides the network-specific context needed for mapping cyber threats to specific network environments, reducing false alarms, and suggesting optimal

FIG. 1 CyGraph knowledge stack.

attack responses. It helps prioritize exposed vulnerabilities, alone and in combination, with focus on protecting mission-critical assets against potential threat sources. It also provides the context for correlating intrusion alerts and other kinds of network events, matching them to known vulnerability paths. This in turn suggests best courses of action for responding to attacks. Specifically, for postattack situational awareness, CyGraph shows possible paths leading up to the current attack locus (backward looking) as well as potential paths for the attacker to advance the attack (forward looking). It also provides a comprehensive framework for computing a variety of metrics for tracking security readiness over time.

CyGraph provides comprehensive query capabilities over its graph knowledge base, including a query language specific to its knowledge domain. This supports a range of cyber analysis tasks, such as mapping an attacker's potential reach and combining isolated alerts into coordinated multistep attack campaigns. CyGraph also provides a variety of interactive visualization capabilities for portraying complex graph query results.

Section 2 discusses previous work related to the CyGraph system. Section 3 then describes CyGraph in more detail. In Section 4, we examine a number of example applications of CyGraph. Section 5 then summarizes this chapter.

2 RELATED WORK

Traditionally, the development of cybersecurity models and analytics has been hampered by a lack of information sharing. MITRE's *Making Security Measurable* (Martin, 2008; The MITRE Corporation, 2013) is a collection of collaborative initiatives for shared information, languages, and processes

for cybersecurity that has helped in that regard. These initiatives encompass software assurance, threat analysis, vulnerability management, malware protection, intrusion detection, incident coordination, and other areas of security. Collaborators include teams from US Department of Homeland Security (DHS), Defense Information Systems Agency (DISA), National Institute of Standards and Technology (NIST), Internet Engineering Task Force (IETF), and many others from government, academia, and industry (The MITRE Corporation, 2014). These standardization efforts facilitate information sharing needed for building CyGraph knowledge graphs. Also helpful are a variety of ontologies and taxonomies that have been proposed for the cybersecurity domain, e.g., Iannacone et al. (2014).

On the product side, various tools for attack graph analysis are available, both Government Off-The-Shelf (GOTS) and Commercial Off-The-Shelf (COTS). GOTS examples include TVA (Jajodia et al., 2005) (commercialized as Cauldron (NSA, 2009)) and NetSPA (Artz, 2002) (commercialized as CyberAnalytix, 2008). Other COTS tools include Skybox (2016) and RedSeal (2016). Attack graphs have also been generated by logic programming (Ou et al., 2005) and model checking (Sheyner and Wing, 2004). More comprehensive reviews of previous work in attack graphs and other graph-based cybersecurity models are given in Lippmann and Ingols (2005) and Schweitzer (2013).

Previous approaches and tools for attack graph analysis have generally employed specialized data structures and algorithms designed for solving specific problems, e.g., attack reachability (Ingols et al., 2006), data aggregation (Noel and Jajodia, 2009a), network hardening (Albanese et al., 2012), sensor placement (Noel and Jajodia, 2007), alert correlation (Ning and Xu, 2004), security posture metrics (Noel and Jajodia, 2014), or risk of unknown (zero-day) vulnerabilities (Wang et al., 2013). Flexibility and extensibility in the face of evolving network environments and adversary threats have not been first-class design criteria. For example, TVA/Cauldron lacks a database persistence layer, and changing the model (new vulnerability scans, firewall rule changes, etc.) requires the entire attack graph to be rebuilt. Such tools are usually implemented with custom code that is difficult to extend as new data sources, model abstractions, analytic techniques, and visualization capabilities are introduced.

Relational database representations have been proposed for attack graphs (Wang et al., 2006). While this has the advantage of a standard model for data representation and queries, the relational model is not the best match for graph problems, especially for evolving network environments and analytic requirements. Extending a relational model requires schema redesign, database reloading, etc. Many graph operations are difficult to express in Structured Query Language (SQL). Moreover, graph traversal in relational databases requires computationally expensive self-join operations.

A class of NoSQL databases known as graph databases has emerged, which are optimized for graph operations. In CyGraph, we employ the Neo4j

graph database (Neo4j, 2016). Neo4j represents node adjacency via direct pointers, which avoids expensive join operations for graph traversal. Neo4j has demonstrated graph traversal performance orders of magnitude better than relational databases (Baas, 2012; Batra and Tyagi, 2012). Query execution times depend only on the size of the traversed subgraph, independent of the size of the overall graph (Robinson et al., 2015).

Many of the problems in big data are amenable to established methods of high-performance computing (HPC), so that boundaries between big data and HPC are blurring (Nadkarni and Vesset, 2014). Graph data structures lack the spatial locality implicit in traditional HPC architectures (e.g., for data arrays). To address this, Cray has developed the Urika-GD™ appliance (Cray, 2014), which has specialized hardware and software for high-performance large-scale graph analytics.

3 DESCRIPTION OF CyGraph

CyGraph is a comprehensive, scalable, high-performance system for analyzing and reasoning about network attack relationships. It correlates data from numerous sources (topology, vulnerabilities, client/server configurations, firewall rules, events, etc.) into a common, normalized model and builds a persistent graph data store representing network attack relationships and associated network data. CyGraph supports queries that identify key vulnerabilities, suggest optimal mitigation strategies, map host-to-host trust relationships, show downstream/upstream paths for attack response, etc. The system includes components to compute analytical graph-theoretic measures such as centrality, degree, connectivity, and diameter. It also provides interactive visualization capabilities for conveying complex dependency relationships.

Section 3.1 describes the overall architecture of the CyGraph tool. Section 3.2 explains in detail about the variety of data sources that CyGraph ingests for building its cyber graph models. Section 3.3 describes how CyGraph leverages NoSQL graph databases within an integrated environment for big data analytics, for synthesizing its graph knowledge base from raw data. Section 3.4 examines details of CyGraph's distributed client–server implementation. In Section 3.5, we introduce a domain-specific query language for CyGraph, which provides a level of abstraction that hides lower-level details of the underlying CyGraph data model. Section 3.6 examines a variety of interactive visualization capabilities in CyGraph that help analysts better understand and communicate knowledge base query results.

3.1 CyGraph Architecture

Fig. 2 is a high-level view of the CyGraph architecture. The architecture includes REpresentational State Transfer (REST) web services for ingest, transform, and analytics (queries and visualization). CyGraph ingests data

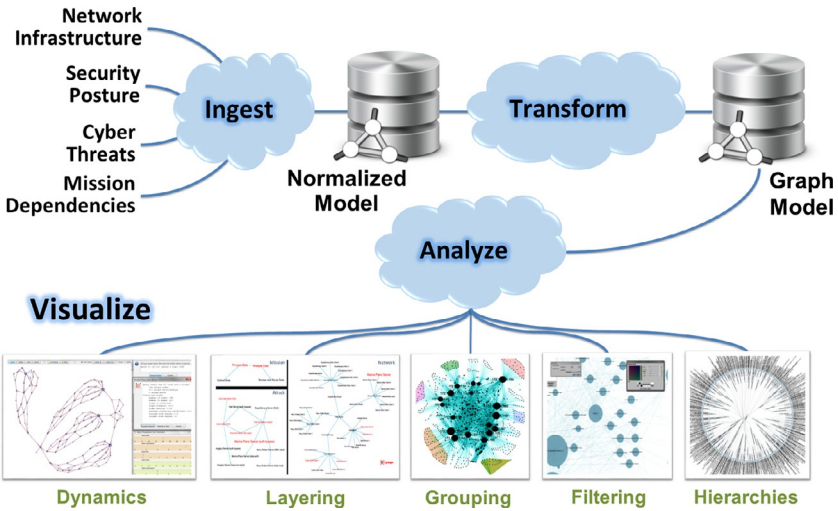

FIG. 2 CyGraph architecture.

from a variety of sources at all layers of its cyber knowledge stack, mapping source-specific data to a common normalized data model. It then transforms the isolated elements of the normalized model into a graph model that captures relevant relationships for the cybersecurity and mission dependence domains. CyGraph also provides a variety of client-side analytic and visual capabilities, including graph dynamics, layering, grouping, filtering, and hierarchical views.

In the CyGraph architecture, the cybersecurity model schema is free to evolve with the available data sources and desired analytics, rather than being fixed at design time. The data model is based on a flexible property-graph formulation implemented in Neo4j. Model extensions are simply the creation of additional of nodes, relationships, and properties in the property-graph data model and require no schema changes or other database renormalizing. Graph pattern-matching queries are expressed in either native Neoj4 query language (Cypher) or our domain-specific CyGraph Query Language (CyQL), which CyGraph compiles to native Cypher.

3.2 CyGraph Data Sources

In the CyGraph architecture, the ingest service provides a standard format for input data, which is processed by source-specific adapters. Thus, data "in the wild" are mapped to the layered CyGraph data model. The network infrastructure layer of this model captures the configuration and policy aspects of the network environment, which forms the basis for modeling security posture (potential vulnerability paths). The cyber threats layer captures events and indicators of actual cyberattacks, which are correlated with elements at the

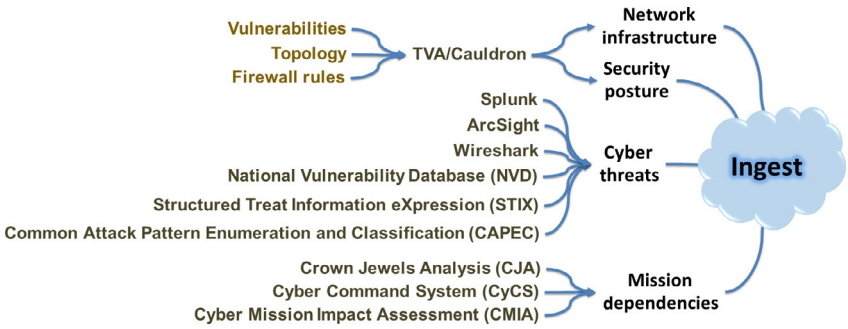

FIG. 3 Example sources for CyGraph data ingest.

lower levels, providing context for cyber events and supporting predictions of subsequent attack spread. The mission dependencies layer shows how cyber activities relate to mission elements.

Fig. 3 shows example data sources for building CyGraph models. These provide elements of CyGraph's cybersecurity/mission knowledge stack. In this way, CyGraph leverages existing tools and data sources for building its rich knowledge graph.

First we consider data sources pertaining to network infrastructure and cyber posture, i.e., network topology, firewall rules, and host vulnerabilities. In CyGraph, we leverage TVA/Cauldron (Jajodia et al., 2011; Noel and Jajodia, 2009b; Noel et al., 2002, 2009; O'Hare et al., 2008), a tool developed at George Mason University for building and analyzing network attack graphs. TVA/Cauldron imports scan results from various vulnerability scanner products. It also parses firewall rules (access control lists) from various firewall vendors. It then analyzes host vulnerabilities, firewall rules, and network topology (subnets, routes, and firewall locations) to enumerate attacker reachability to vulnerable hosts. This in turn provides a model for network infrastructure and security posture in CyGraph. The security posture layer of CyGraph's knowledge stack supports prioritization and optimization of proactive security measures in advance of attack. This layer also provides context for responding to attacks.

In addition to source data about network infrastructure and cyber posture, CyGraph ingests various data sources for cyber threats, both potential (threat intelligence) and actual (cyberattack events). The Splunk log analysis tool (Zadrozny and Kodali, 2013) indexes data from network- and host-based sensors, e.g., intrusion detection systems and other specialized tools providing live threat indicators. CyGraph also processes packet capture data via Wireshark (Sanders, 2011), e.g., for analyzing general traffic patterns. Cyber threat intelligence sources for CyGraph include the National Vulnerability Database (NVD) (2016), Structured Threat Information eXpression (STIX™) (STIX, 2016), and Common Attack Pattern Enumeration and Classification

(CAPEC™) (The MITRE Corporation, 2016). Threat Assessment and Remediation Analysis (TARA) (Wynn et al., 2011) provides a structured methodology for elements of both cyber posture and threats.

Through its mission dependencies layer, CyGraph supports analysis of potential and actual impact of cyberattacks and defenses on organizational functions (missions). To populate this layer, CyGraph leverages established methodologies and tools (Noel and Heinbockel, 2015). This includes Crown Jewels Analysis (CJA) (The MITRE Corporation, 2009), which is a structured methodology for identifying mission-critical cyber assets. It also includes Cyber Command System (CyCS) (The MITRE Corporation, 2016), a tool that captures hierarchical dependencies among mission components, and maps mission operations to the network operations that support them. CyGraph can also leverage Cyber Mission Impact Assessment (CMIA) (Musman et al., 2011), which evaluates time-dependent effects of cyberattacks on mission effectiveness and performance.

In the CyGraph framework, the data model is schema free, so that the model is decoupled from the storage implementation. This provides flexibility in data sources, and how the data are transformed (cast as a graph) determines a particular instantiated CyGraph model. User queries must match a given instantiation. This means that there is a knowledge engineering phase in developing source-specific adapters for populating CyGraph instances, and for formulating relevant queries according to the graph model.

3.3 Big Data Analytics in CyGraph

Big data analytics involve a process of continual discovery. CyGraph discovers interrelationships relevant to attacker progress through a network and corresponding mission impact. This includes vulnerabilities in the usual sense, as well as other attack relationships that enable attacker progress, such as remote desktop and stored credentials. Network events (alerts, flows, etc.) are mapped to these attack relationships, providing context for correlating otherwise isolated events. This shows the next steps that an adversary can take, for optimal response by defenders.

This kind of complex, interconnected, unpredictable data is best captured in a graph model (data structure). Relational databases work well for referencing discrete data items and fixed relationship patterns, e.g., bank customers and their accounts. But the relational model has difficulties when the relationships themselves are variable, especially as in cyber security.

Graph databases are a class of NoSQL database (Gudivada et al., 2016) that embraces graphs as the underlying model for data representation and storage (Angles and Gutierrez, 2008). They often employ semantic (pattern-matching) query languages, which allows retrieval of both explicitly defined information as well as information that can be implied (e.g., through graph traversal). They are applicable when the information about data relationships

is as important (or even more important) as the data themselves. A survey of popularity trends for various classes of databases (Andlinger, 2015) shows that graph databases have had a fivefold increase in popularity over the last 2 years (ending March 2015), a much higher growth rate than any other class of database.

Within the graph database class, Neo4j dominates popularity (DB-Engines, 2015). Neo4j is open source, with commercial licensing for an enterprise version also available (Neo4j, 2016). Unlike most NoSQL databases, Neo4j enforces ACID (atomicity, consistency, isolation, durability) transaction properties usually associated with relational database systems.

Graph databases represent node adjacency via direct pointers. This avoids expensive join operations or other index lookups for graph traversal. Graph databases have been shown to be orders of magnitude faster than relational databases for graph traversal, especially deeper traversals (Vukotic et al., 2015). In Neo4j, graph traversal speed depends only on the size of the query result actually traversed, independent of the total size of the graph.

CyGraph is deployed in the MITRE Cyber Analytic Virtual Environment (CAVE), shown in Fig. 4. CAVE provides an integrated, scalable, fault-tolerant, and managed virtual environment for big data analytics. It hosts a suite of cyber data repositories, knowledge bases, and analysis engines, as well as capabilities for querying and visualization.

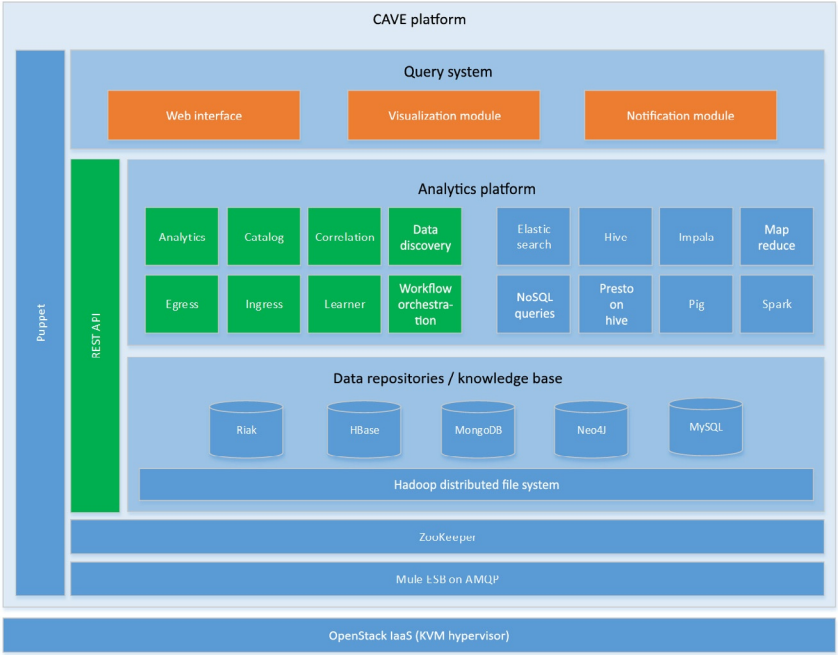

FIG. 4 CAVE stack for big data analytics.

CyGraph synthesizes new network, cyber, and mission knowledge, which is stored in CAVE's knowledge base. In the analytics layer, CyGraph provides graph-based data cataloging, correlation, analytics, and queries. At the top of this big data analytics stack, CyGraph provides novel forms of interactive visualization.

3.4 CyGraph Client–Server

The CyGraph architecture is based on a distributed client–server model, partitioning tasks between service providers (servers) and service requesters (clients). CyGraph clients and servers can be deployed on separate hosts (real and/or virtual) or can be configured to run on a single (local) host. Typically, CyGraph services (middle-tier server and backend database) are implemented as separate virtual machine instances, i.e., in VMware (2016). Such a VMware instance can also be migrated to Amazon Elastic Compute Cloud (Amazon EC2) (Amazon Web Services, 2016) for web-scale cloud computing.

Here are the functional roles for each tier of the CyGraph client–server architecture:

CyGraph Client: The CyGraph Client is a graphical user interface for posing CyGraph queries and visualizing query results. The client communicates with the CyGraph Server through RESTful web application program interface (API) calls. The predominant CyGraph Client is implemented in Java as a desktop application. Because of the decoupled client–server architecture, other clients can be implemented on other platforms, e.g., a web browser.

CyGraph Server: The CyGraph Server acts as a middle-tier intermediary between the CyGraph Client and the CyGraph Database. It provides a layer of abstraction that gives a common service interface, regardless of the how the database backend (and its native query language) is implemented. The server handles the interpretation of CyGraph domain-specific query language (CyQL) into native database queries. The server also houses a library of commonly issued queries, to capture domain knowledge, streamline the analytic workflow, and help ease the learning curve for new analysts.

CyGraph Database: The CyGraph Database stores the graph data (nodes, relationships, and properties). It processes queries from CyGraph Server, in the native language of the database implementation (Neo4j).

Fig. 5 shows the key components for the implementation of each tier of the CyGraph client–server architecture. A key library for CyGraph Client is GraphStream (Dutot et al., 2007), which provides basic functions for graph visualization, styling, and user interaction. CyGraph Server leverages the Spring web model-view-controller (MVC) framework (Yates et al., 2006), which separates the representation of information from how that information

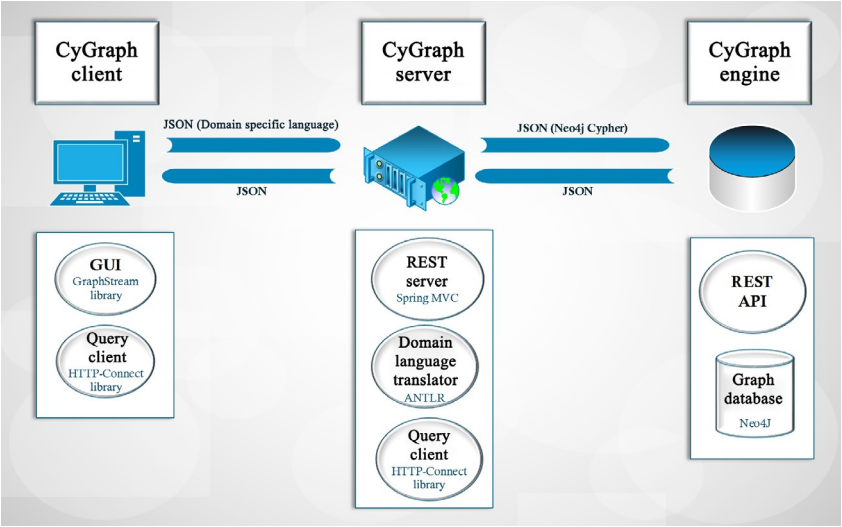

FIG. 5 Components of CyGraph client–server architecture.

is presented to the user. CyGraph Server also applies ANTLR (ANother Tool for Language Recognition) (Parr, 2013) for translating queries in the CyGraph domain-specific query language (CyQL) to corresponding queries in a native graph database query language. CyGraph Client and CyGraph Server also leverage Jersey RESTful web services (Gulabani, 2014) for connecting to CyGraph services via Hypertext Transfer Protocol (HTTP).

In the CyGraph architecture, the analyst formulates a graph pattern-matching query in CyGraph Client (in either domain-specific CyQL or native Neo4j Cypher query language) and submits it. CyGraph Client wraps the query in a JSON message and sends it to the corresponding service (CyQL or Cypher) on CyGraph Server. CyGraph Server processes the query and sends it to CyGraph Database. For a CyQL query, CyGraph Server translates (compiles) it to the corresponding Cypher query; otherwise, it just forwards the native Cypher query.

In response, CyGraph Database (backed by Neo4j) executes the Cypher query and returns the results (matched subgraph) to CyGraph Server as Neo4j-formatted JSON. CyGraph Server parses the Neo4j JSON, builds the corresponding vendor-neutral CyGraph model JSON, and sends that to CyGraph Client for rendering as dynamic graph visualization.

The loosely coupled modular design of CyGraph makes it straightforward to develop alternative implementations for the various tiers of the client–server architecture. For example, CyGraph clients provide alternative graph visualization capabilities in web browsers through standards such as HTML5, JavaScript, Cascading Style Sheets (CSS), and Scalable Vector Graphics (SVG), by leveraging D3.js (Bostock et al., 2011), vis.js (vis.js, 2016), and FoamTree (Carrot Search, 2016). Client-side code also exports CyGraph models into Graphviz.

3.5 CyQL: CyGraph Domain-Specific Query Language

Graph queries in CyGraph are specifications for matching subgraph patterns of interest, written in declarative query language (Harper, 2013). In declarative (nonprocedural) languages, one specifies what needs to be done (i.e., match a particular graph pattern) rather than exactly how to do it (checking properties, traversing edges, etc.).

We provide an additional layer of abstraction by defining a domain-specific language (DSL) for the CyGraph data model, which we call *CyQL*. There are a number of advantages for doing this. A DSL increases the clarity of analytic queries against our cyber data model, especially as the model becomes more complex. CyQL does this by encoding cyber semantics into the query language itself, encapsulating and hiding many of the constraints that must be expressed in the native graph database queries.

While a general-purpose graph query language is broadly applicable across all domains, CyQL is specialized to the application domain within the scope of CyGraph. This in turn helps reduce the learning curve and increase the productivity of security analysts and content developers using CyGraph.

The additional layer of abstraction provided by CyQL also allows CyGraph to support multiple backend data engine implementations, each with their own native query language. This means that users and application software that integrate with CyGraph interact with a single (domain-specific) language, which is independent of the particular native query language that implements the DSL.

Section 3.5.1 describes CyQL in more detail. Section 3.5.2 examines a number of example CyQL queries to see the language in practice.

3.5.1 Description of CyQL

Fig. 6 shows an instance of the underlying CyQL data model, expressed as a graph of entity (node) and relationship (edge) types. Here, properties (name–value pairs) for nodes and edges are omitted for clarity. It is interesting that these four data model areas (mission readiness, network infrastructure, cyber threats, and cyber posture) are tied together by only two node types—machines and exploits.

CyGraph (Server) compiles CyQL (DSL) queries to the native language for our graph database implementation (Cypher for Neo4j). The CyQL lexical analyzer and parser are generated via ANTLR (Parr, 2013). The lexical analysis groups input CyQL queries into tokens. The parser recognizes the tokens in terms of CyQL's grammar structure and maps them to a parse tree. CyGraph Server then iterates over the parse tree and generates corresponding Cypher query code.

ANTLR defines notation for specifying an input language's grammar, in Extended Backus–Naur Form (Information Technology, 1996). We thus define the grammar rules for CyQL. In CyQL, each function calls returns a

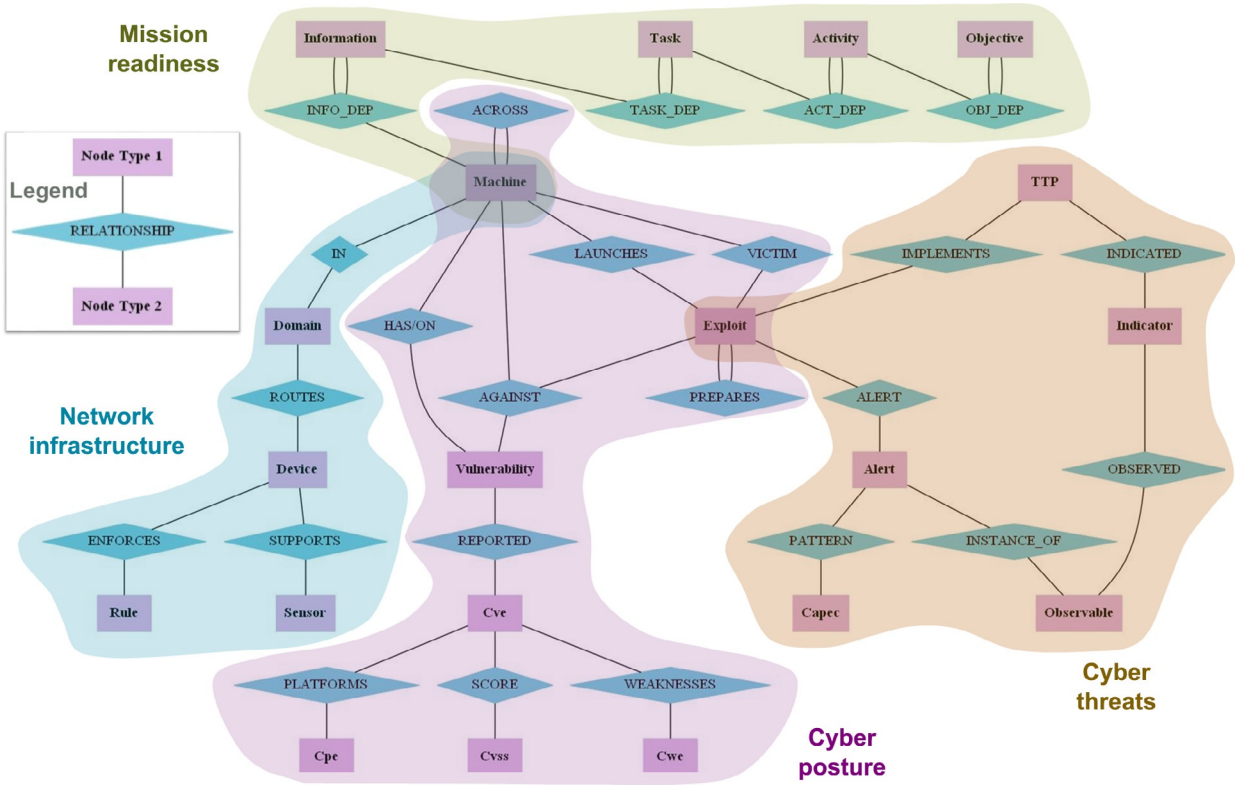

FIG. 6 Knowledge graph model for CyQL.

```
grammar CyQL;
@header {
    package org.mitre.cygraph.dsl;
}

query : subset (SETOP subset)* EOF ;
subset : call
           | '(' subset (SETOP subset)* ')' ;
call : FUNCTION '(' ')'
     | FUNCTION '(' paramList ')' ;

paramList : param (',' param)* ;
param : ID '=' disjunction
      | ID '=' value
      | ID '=' valueList ;

valueList : '[' value (',' value)* ']' ;
value : ID
      | INT
      | GLOB
      | STRING
      | IPV4
      | ipv4Range ;

disjunction : conjunction ('or' conjunction)* ;
conjunction : specifier ('and' specifier)* ;
specifier : '{' paramList '}'
          | '(' disjunction ')' ;

ipv4Range : IPV4 '-' IPV4
          | IPV4 '/' IPV4
          | IPV4 '/' INT ;

IPV4 : INT '.' INT '.' INT '.' INT ;
```

```
FUNCTION : 'network' | 'all' |
'exploitPaths' | 'correlateAlerts' |
'cveData' ;
SETOP : 'join' | 'except' | 'union' |
'intersect' ;
ID : [a-z][a-zA-Z0-9]* ;
GLOB : [a-zA-Z*][a-zA-Z0-9*_-]* ;
INT : [0-9]+ ;

STRING : '"'
            (   ESC
                |   ~('"'|'\\')
                )*
            '"' {
                setText(

org.antlr.v4.misc.CharSupport
.getStringFromGrammarStringLiteral(
   getText())
   );
}
;

fragment ESC :   '\\' (["\\/bfnrt] |
UNICODE) ;
fragment UNICODE : 'u' HEX HEX HEX HEX;
fragment HEX : [0-9a-fA-F];

COMMENT : '//' ~[\n]* '\n' -> skip;

BLOCK_COMMENT : '/*' .*? '*/' -> skip;
WS : [ \t\r\n]+ -> skip ;
```

FIG. 7 Grammar for prototype version of CyQL (CyGraph Query Language).

matching subgraph. Optionally, we can chain multiple function calls with the keyword "join." Each function (optionally) takes a list of parameters (name–value pairs). A value can be a single key-value pair with a primitive-type value, a list of literal values for a parameter (key), or a "specifier," which is a list of parameters and their values (surrounded by curly braces). Function calls cannot be nested, but specifiers can.

Fig. 7 shows the ANTLR grammar for CyQL. It defines a query as a collection of function calls (query types), such as mapping a network topology, correlating intrusion alerts, or traversing exploitable vulnerability paths. Function parameters can be used with Boolean operations (AND, OR, and NOT) to refine a query based on node properties and relationships. These results can then be expanded by JOINing other functions, to provide additional context. The CyQL grammar also defines rules for parsing cyber entities such as IP addresses, address ranges, and host names (including support for wildcard expressions).

3.5.2 *Example CyQL Queries*

As an example, consider a CyQL query for finding "exploit paths," i.e., sequences of vulnerabilities that an adversary could exploit for lateral movement through a network. These need not be simply linear chains. Rather, the query finds all possible paths (including branches/merges) between a specified set of machines. From a security analyst's perspective, this means "show me how an adversary can get from these machines to those machines."

While that kind of query answers the basic question about attacker reachability among network machines (exploitable paths), the analyst might want to refine the query further to provide more specific focus. For example, we might want to only include a certain group of machines (e.g., having a common hostname pattern) or include only those machines that have alerts for them.

On the other hand, the analyst might wish to expand a basic query by including additional information such as vulnerabilities associated with the machines in a set of exploitable paths. The query language should allow a rule as simple as *join vulnerabilities*() such that the appropriate vulnerability subgraphs are joined with the corresponding machine nodes of the exploit-paths subgraph.

The function *exploitPaths()* determines the structure and the types of edges in the matching subgraph. Its arguments get compiled to an SQL-like WHERE clause that constrains the node properties that match the query. For the *exploitPaths()* function, constraining arguments include starting and ending machines for an exploitable subgraph.

Consider this example CyQL query:

```
exploitPaths(start = ({subnet=1.1.3.0/24} or
             {ip=[1.1.4.32, 1.1.4.33]}) and
             {hostname=*-VM*}, end = {name = "DB Server"})
```

Here is the resulting query compiled from CyQL to Neo4j Cypher:

```
MATCH (start)-[r:AGAINST|VICTIM|ON|LAUNCHES|IN|ROUTES*]-(end)
WHERE ((start.subnet = "1.1.3.0/24" OR
             start.ip IN ["1.1.4.32", "1.1.4.33"])
      AND start.hostname =~ "^[a-zA-Z0-9_-]*-VM-[a-zA-Z0-9_-]*$")
AND (end.name = "DB Server")
RETURN start, r, end
```

The CyQL version of this query is much less verbose. It encapsulates the knowledge of allowed relationship types (AGAINST, VICTIM, ON, LAUNCHES, IN, and ROUTES) for subgraphs representing attack reachability between machines. This is based on the role of those particular relationship types in the CyGraph data model.

Fig. 8 is the resulting parse tree for this CyQL query. It shows the starting machines as a disjunction (OR) of the specified IP addresses, combined conjunctively (AND) with a wildcard expression for host names. The ending machine is simply matches a particular host name.

Fig. 9 shows how this query formulation, translation, and execution happens within the overall CyGraph architecture. The analyst formulates a query expressed in CyQL, which the client submits to the CyGraph Server. The service parses the CyQL query, verifies the types of function parameters, translates the query to equivalent Neo4j Cypher, and submits the resulting Cypher query to the database engine.

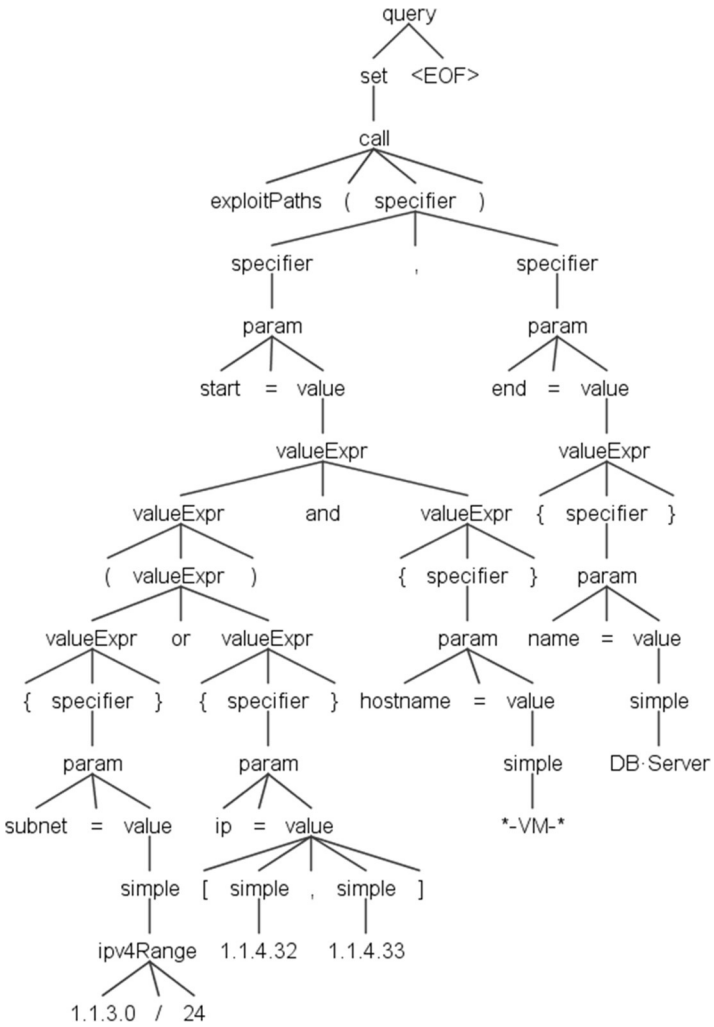

FIG. 8 Grammar parse tree, for example, CyQL query.

CyQL's "filter, then expand" model allows for flexible queries while maintaining a simple and readable syntax. As an example, consider a simple network topology query:

```
network()
```

Here is the result of compiling this CyQL query into Cypher:

```
MATCH (domain:Domain)-[r:ROUTES*]-(device:Device)
RETURN domain, r, device
```

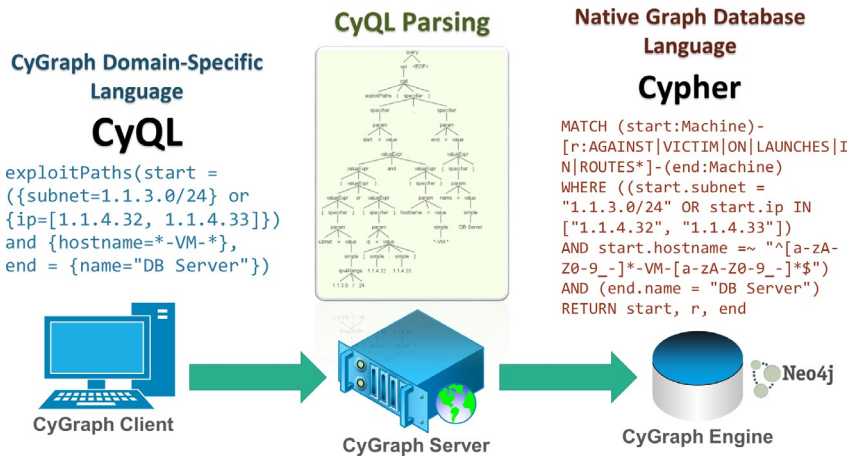

FIG. 9 CyQL domain-specific query language processing in CyGraph.

This query returns only the backbone of the network—the protection domains and devices such as routers and switches. Endpoint machines are not included in the resulting subgraph, but can easily be added:

```
network() join machines()
```

This compiles to:

```
MATCH (domain:Domain)-[r:ROUTES*]-(device:Device)
OPTIONAL MATCH (machine:Machine)
RETURN r, machine, domain, device
```

The *OPTIONAL MATCH* clause in Cypher functions similar to a *LEFT JOIN* in SQL. This query returns the network backbone and all machines on the network.

An analyst may want to consider only a subset of machines. This is done via parameters to the *machines()* function:

```
network() join machines(ip = 1.1.3.2)
```

```
MATCH (domain:Domain)-[r:ROUTES*]-(device:Device)
OPTIONAL MATCH (machine:Machine)
      WHERE machine.ip = "1.1.3.2"
RETURN r, machine, domain, device
```

Parameters can refer to properties on the machine nodes, or on related objects. For instance, we can include only the machines in certain domains:

```
network() join machines(domain = {name = DMZ}
      or {name = "Data Center"})
```

```
MATCH (domain:Domain)-[r:ROUTES*]-(device:Device)
OPTIONAL MATCH (machine:Machine), (machine)-[:IN]->(domain2)
      WHERE (domain2.name = "DMZ") OR (domain2.name = "Data Center")
RETURN r, machine, domain, device
```

Alternatively, we might be interested in machines with known vulnerabilities:

```
network() join machines(vulnerable = true)
```

```
MATCH (domain:Domain)-[r:ROUTES*]-(device:Device)
OPTIONAL MATCH (machine:Machine)
      WHERE (:Vulnerability)-[:ON]->(machine)
RETURN r, machine, domain, device
```

This query shows vulnerable machines on the network but does not include nodes for the vulnerabilities themselves. To add them, we join another function:

```
network() join machines(vulnerable = true) join vulnerabilities()
```

```
MATCH (domain:Domain)-[r:ROUTES*]-(device:Device)
OPTIONAL MATCH (machine:Machine)
      WHERE (:Vulnerability)-[:ON]->(machine)
OPTIONAL MATCH (vulnerability)-[:ON]->(machine)
RETURN r, machine, domain, device, vulnerability
```

Another way to expand a query is with the "!" operator. By default, CyQL functions return the smallest subgraph that makes sense. For instance, the *machines*() function returns only machine nodes, even if other types of nodes are used in the search parameters. Adding a "!" to the end of a function name makes it return all the nodes and relationships used in the query. As an example, consider this query (which has no "!" operator):

```
machines(domain = {name=DMZ}, vulnerabilities = {name=Heartbleed})
```

This returns machines in the "demilitarized zone" (DMZ) that are vulnerable to Heartbleed, but does not return the domain or vulnerability nodes. Alternatively, this version of the query includes the "!" operator:

```
machines!(domain = {name=DMZ}, vulnerabilities = {name=Heartbleed})
```

This returns the same machines as the first version, plus the DMZ domain node, the Heartbleed vulnerability node, and the relationships that connect them to the machines.

There is a difference between

```
machines!(vulnerabilities = {name = Heartbleed})
```

and

```
machines(vulnerabilities = {name = Heartbleed}) join vulnerabilities()
```

in that the first query will return only one vulnerability node for Heartbleed, while the second will include any additional vulnerabilities on Heartbleed-vulnerable machines.

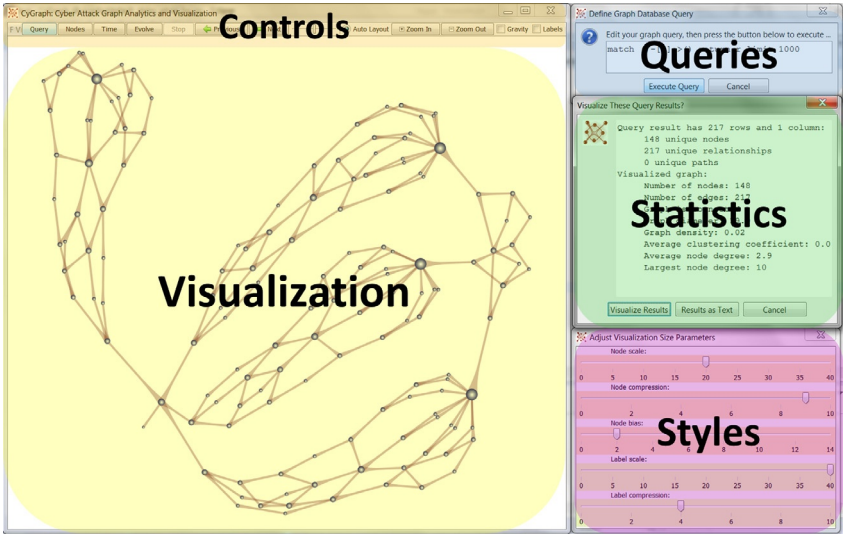

FIG. 10 CyGraph client-side user interface.

3.6 CyGraph Interactive Visualization

In the CyGraph architecture, client applications submit queries to the CyGraph service and process the query results. This decoupled architecture supports various clients with a variety of interactive modalities, including ad hoc queries, query expansion/pivoting/filtering, visual summarization, and dynamic graph evolution over time.

Fig. 10 shows one such CyGraph analysis/visualization client. This client allows an analyst to pose graph pattern-matching queries. The client then renders the query result (instances of a matched pattern) through interactive graph visualization. The client also has functions for graph statistics, styling, spatial layout, and evolution over time.

In general, adding constraints to queries (more specific patterns) yields smaller matched subgraphs. An analytic strategy is to begin with more general queries and then refine them as more is learned. This helps focus the analysis, manage complexity, and improve performance.

As an example, a query against a particular graph knowledge base yields a subgraph of 200,000 vertices and 400,000 edges, as shown on the *left side* of Fig. 11. The query itself completes in only a few seconds, but the vertex positioning needed for effective visualization (as shown in Fig. 12) takes about half an hour.[a] Using the same knowledge base, a more constrained graph

[a]This is for a Dell Latitude E6530 laptop running 64-bit Windows 7 Enterprise, with Intel Core i7-3720QM @ 2.60 GHz (8 logical cores, 4 physical and 2 logical per physical), and 8 GB of memory.

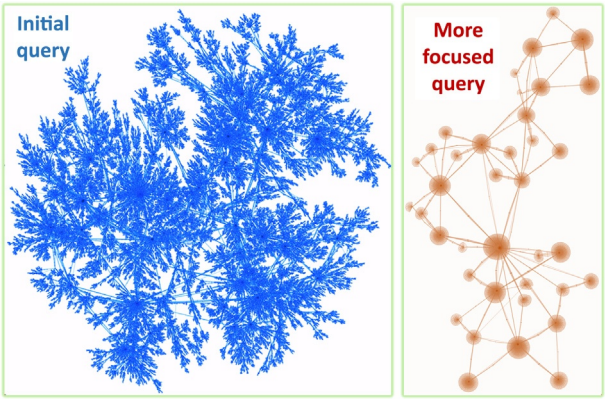

FIG. 11 Constraining CyGraph query results to subgraph of interest.

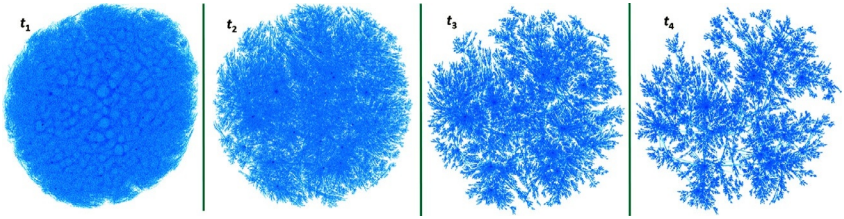

FIG. 12 Progression of graph visualization layout.

query yields a matching subgraph of 10,000 nodes and 20,000 edges, as shown in the right side of Fig. 11. In this case, graph visualization layout completes in only a few seconds.

CyGraph clients provide a variety of ways for interacting with graph query results. This includes filters on node and edge property values, to focus graph visualization (matched queries) on selected criteria. Property-dependent styling choices (e.g., node/edge size, color, transparency, edge routing, and arrow shapes) can also be defined. For example, Fig. 13 shows configuration dialogs for defining filter properties and background color.

CyGraph also supports visual clustering of graph nodes, based on property values or through manual selection. This is illustrated in Fig. 14.

4 EXAMPLE APPLICATIONS

Rather than relying on fixed analytics and visualizations, CyGraph gives the analyst the power and flexibility for crafting queries to solve the problem at hand. Complementary to queries that discover patterns of interest, CyGraph's interactive visualization conveys discovered patterns in ways that help induce faster learning and deeper understanding. This section describes a variety of applications that benefit from CyGraph analysis and visualization, for both real data sources as well as driven by simulations.

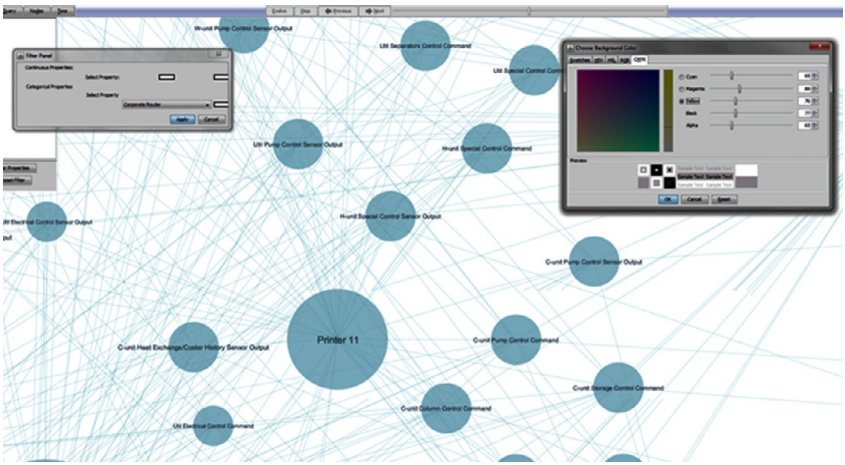

FIG. 13 Defining node filter properties and background color.

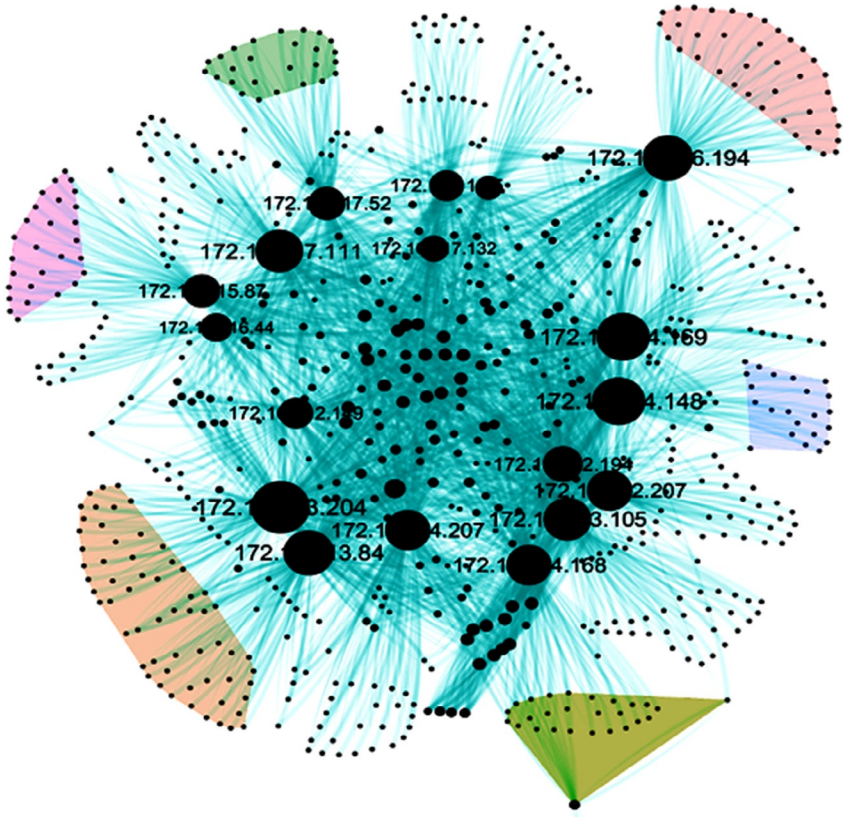

FIG. 14 Clusters of related nodes.

Section 4.1 describes query-driven analytics in CyGraph for extracting relevant portions of the knowledge base for solving particular cyber operational problems. Section 4.2 examines how CyGraph supports cybersecurity modeling and simulation tasks.

4.1 Cyber Analytics

CyGraph supports deep analytics within and across the layers of its knowledge graph stack. This includes network infrastructure that hosts cyber attacks and defenses, cyber posture gained by proactive network hardening in advance of attack, cyber threats (both potential and actual) against the network environment, and associated mission impacts.

Section 4.1.1 describes how CyGraph can leverage the TVA/Cauldron tool to map vulnerability exposures across network infrastructure to assess network security posture. Section 4.1.2 integrates such vulnerable attack paths with cyber threat information and actual alerts. Section 4.1.3 shows how CyGraph can analyze the impact of cyberattacks on an organization's business/mission functions. Section 4.1.4 presents a case study that illustrates a number of CyGraph analytic capabilities.

4.1.1 Network Infrastructure and Cyber Posture

A basic capability for assessing cyber posture is to map vulnerability exposures across network infrastructure. This helps prioritize host vulnerabilities, identify insecure access policy rules, and show how attackers can potentially leverage multiple vulnerabilities to incrementally penetrate a network. As shown in Fig. 15, this analysis requires an enumeration of host vulnerabilities (e.g., from a vulnerability scanning tool), a topology defining network segmentation and location of firewalls, and access rules for the firewalls. For this we can leverage the TVA/Cauldron tool, which analyzes the topology, vulnerabilities, and rules to map paths of exposed vulnerability across a network, known as an attack graph.

Fig. 16 shows the TVA/Cauldron attack graph for a particular network. This has machines grouped into subnets, with edges showing exposed vulnerabilities across subnets. Implicitly, machines within subnets (more generally known as protection domains) have full access to one another's vulnerabilities, i.e., a fully connected subgraph.

In the TVA tool, the attack graph is visualized in a predetermined way, with limited options for constraining the graph, e.g., attack start and goal (highlighted green (gray in the print version) and red (dark gray in the print version)). For managing attack graph complexity, it relies on visual aggregation, such as collapsing protection domains to single nodes to show attack reachability at a domain level and using a single edge to represent the full set of vulnerabilities exploitable from one host to another. In Fig. 16, all protection domains are expanded to show exploitation at the machine level. This visualization is somewhat cluttered, even after judicious manual positioning of machines after expanding subnet boxes, obscuring salient patterns.

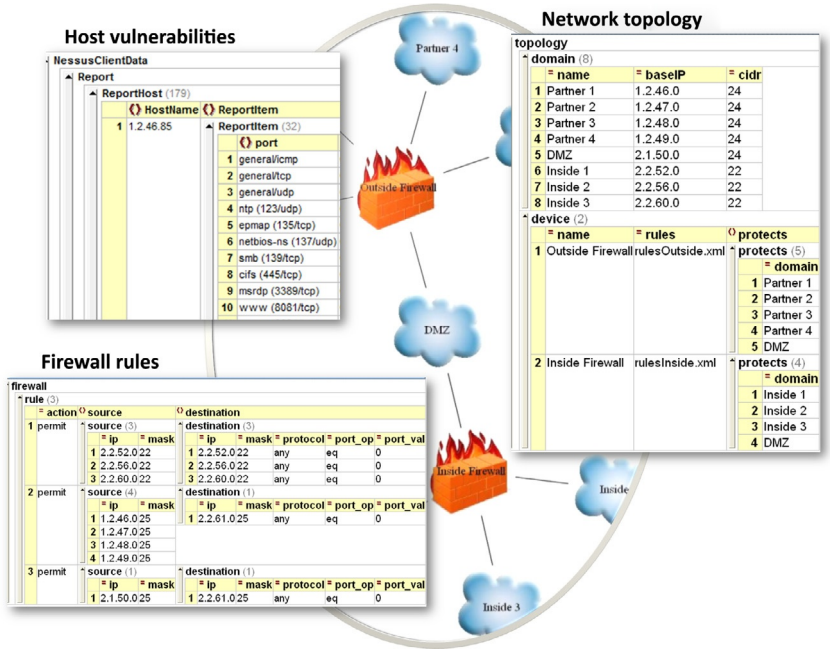

FIG. 15 Network topology, vulnerabilities, and firewall rules.

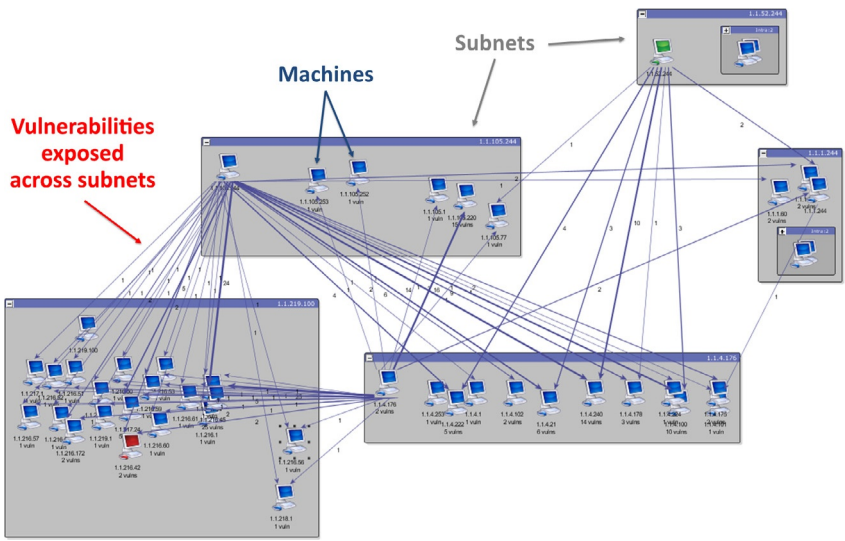

FIG. 16 TVA attack graph visualization.

We ingest this attack graph model into CyGraph, transforming it according to the knowledge graph model of Fig. 6. This has machine, vulnerability, and domain node types, and their corresponding relationship (edge) types. Then for analysis, a naïve initial query (requiring no knowledge of the underlying data model) could be simply "`MATCH ()-[r]->() RETURN r`." This pattern matches all relationships, yielding the full knowledge graph in Fig. 17. Note that this visualization shows all model relationships in full detail, including those that are aggregated or implicit in Fig. 16. Still, some key patterns are apparent, e.g., two machines (1.1.105.244 and 1.1.4.176), and to a lesser degree another one (1.1.52.244) that have the most extensive attack reachability (to exploitable vulnerabilities to other machines).

In CyGraph, queries can be successively refined (constrained) to focus the analytic results. For example, the details of the particular vulnerabilities might be less important than the attack relationships among the machines themselves. In this case, we can pose the query "`MATCH ()-[r:ACROSS|IN]->() RETURN r`." This pattern constrains relationships to only those linking machines across protection domains (ACROSS) and machine memberships within domains (IN). This yields the resulting matched subgraph in Fig. 18. This clearly shows the strong dominance of 1.1.105.244, 1.1.4.176, and 1.1.52.244, along with their reachable machines in common. The domain membership relationships show potential victim machines that are not directly exploitable across domains, but can be exploited with only one additional attack step.

For prioritizing vulnerabilities, an important strategy is to focus on those that are exposed across protection domains (Noel and Jajodia, 2014). In our model, that corresponds to the query "`MATCH ()-[r:ACROSS]->() RETURN r`." This yields the matched subgraph in Fig. 19, which just shows machines that can directly attack across protection domains. This clearly highlights the machines with heavy access to vulnerabilities, and the vulnerable machines that they can reach. In this figure (as for Figs. 17 and 18), edge directionality has arrow heads narrow and arrow tails wide.

This simple example illustrates some important properties of the CyGraph approach. In general, the cyber posture layer can be populated with any relationships capturing potential attacker advantages (at all layers of the network stack; Ritchey et al., 2002), e.g., host inventory agents mapped to reported vulnerabilities (Noel et al., 2009).

In more traditional cybersecurity tools, the analytic and visual behaviors are determined at design time and built into code. In CyGraph, the model structure is driven by the data sources and how they are transformed into a graph knowledge base. Then, given some understanding of the underlying knowledge model, the analyst can construct ad hoc queries to fine-tune analytic results. As we show in subsequent sections, we can enrich the graph knowledge base with additional elements (arbitrary nodes, relationships, and properties) relevant to cybersecurity and mission assurance in our environment. Richer knowledge supports more refined queries and specialized interactive visualizations.

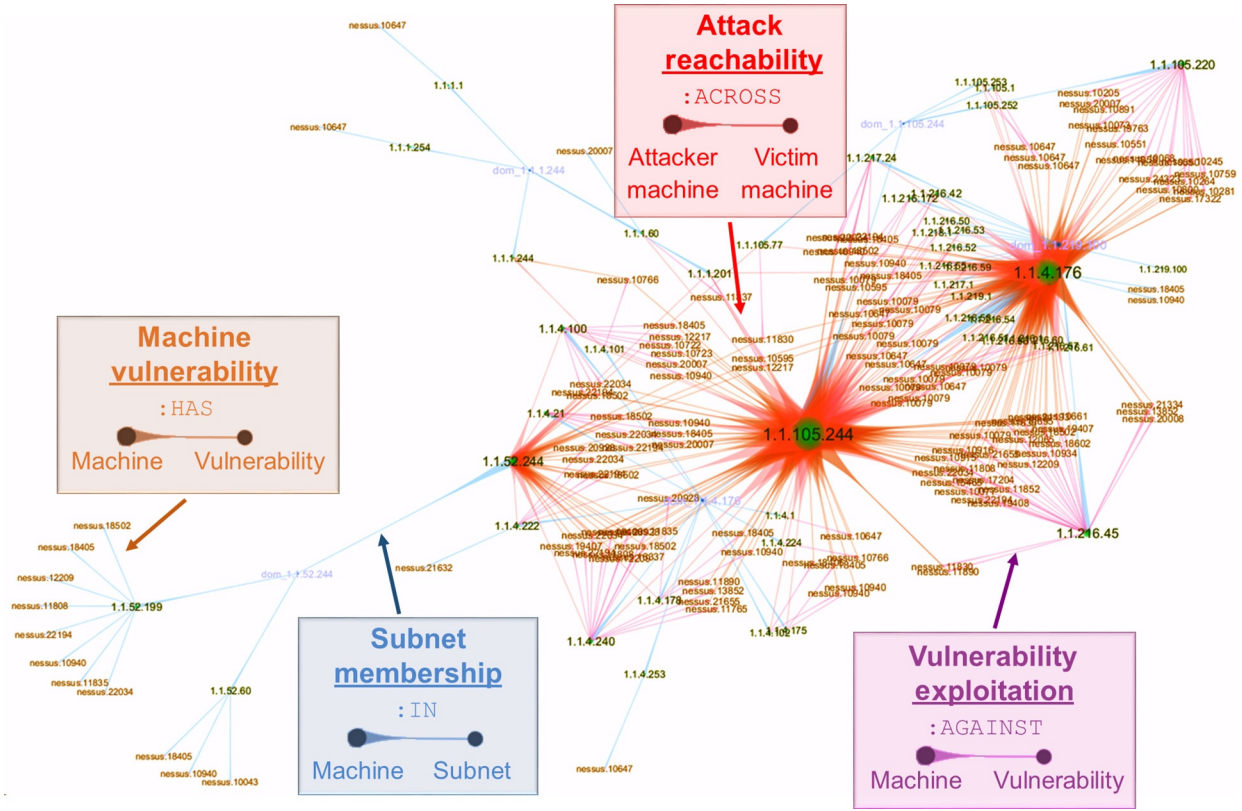

FIG. 17 Full graph of security posture relationships.

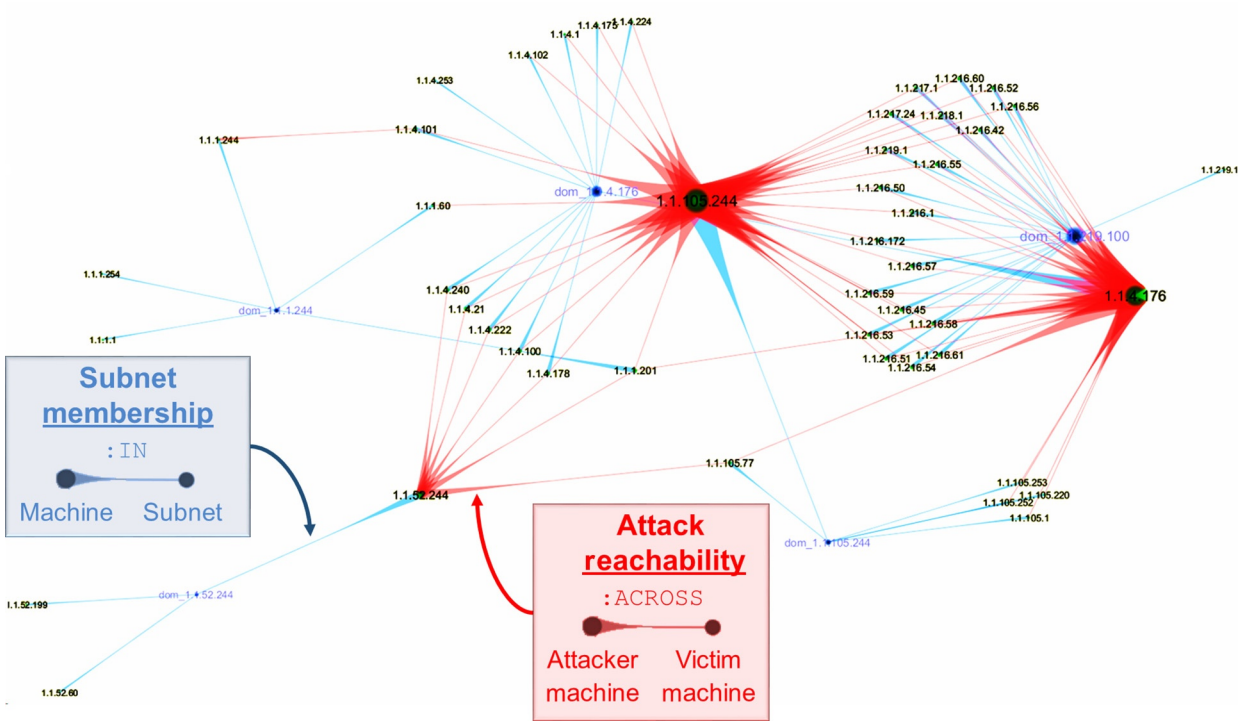

FIG. 18 Machines vulnerable across subnets and subnet memberships.

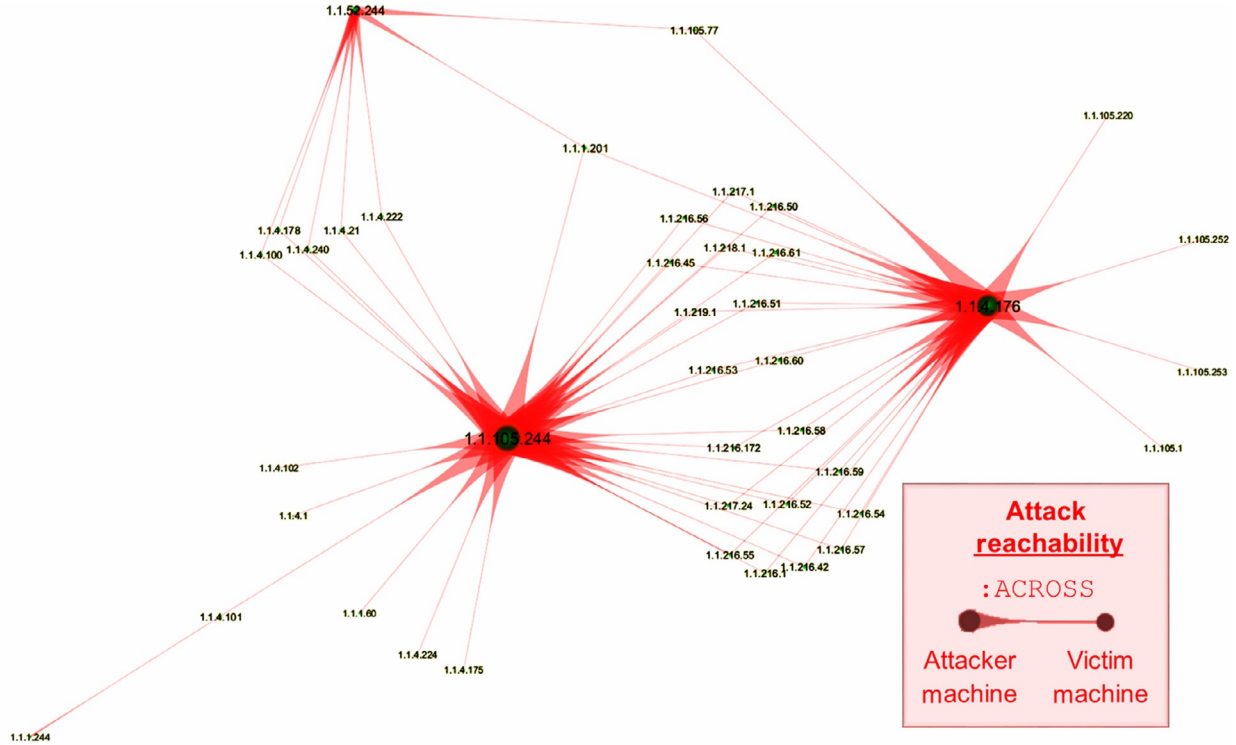

FIG. 19 Machines with vulnerabilities exposed across subnets.

4.1.2 Cyber Threats

In CyGraph, the cyber posture layer enumerates potentially exploitable vulnerabilities within the network infrastructure. This forms the context for understanding potential cyber threats and responding to actual ones. The population of tools such as CyGraph is greatly facilitated by standard languages for sharing threat intelligence, including attack patterns and associated observables.

An important resource for such threat intelligence sharing is the CAPEC™, a standardized catalog and taxonomy of attack patterns. CAPEC provides detailed characterizations of each attack pattern and organizes the patterns into a taxonomic hierarchy (general attack classes, their subclasses, and specific attacks). Navigating CAPEC on the web requires following parent–child hyperlinks embedded in the textual content. This does not lend well to understanding the overall hierarchical taxonomic structure. CyGraph tree visualization capabilities help in this regard, providing a variety of interactive visualization modalities for visualizing and navigating the CAPEC taxonomy (Noel, 2015).

The left side of Fig. 20 shows the CAPEC taxonomy. Here, nodes are attack pattern classes and edges are parent–child relationships (with replicated subtrees for the relatively few instances of classes with multiple parents). The right side of the figure shows one kind of interactive tree visualization employed by CyGraph, which has a Cartesian layout of nodes.

Other forms of tree visualization seek to maximize use of display space, e.g., sunburst and treemap visualizations. A type of treemap visualization in CyGraph with particularly desirable visual properties is the Voronoi treemap (Balzer and Deussen, 2005; Carrot Search, 2016). Fig. 21 shows a Voronoi treemap visualization for the CAPEC taxonomy. The left side of the figure is the initial view, which has the highest-level (most general) attack classes. The analyst can then drill down into successive levels (more specific attack classes). In this visualization, the area of each attack class is a function of the number of its child subclasses, so that more populated parts of the taxonomy are emphasized. Transparency and color muting provide some context of parent and child classes.

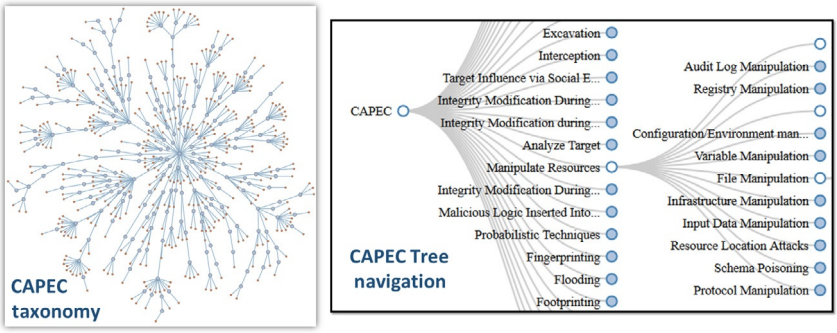

FIG. 20 CAPEC taxonomy of attack patterns.

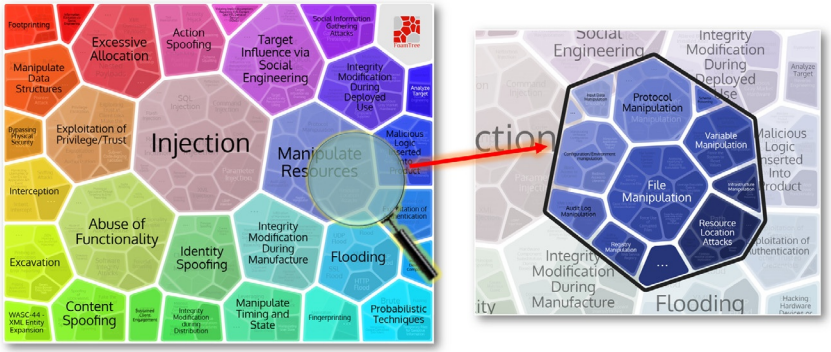

FIG. 21 Interactive Voronoi treemap of CAPEC taxonomy.

In CAPEC, each attack pattern includes the phases of attack (explore, experiment, and exploit) comprising it, including indicators for success/failure of each phase. This provides opportunities for understanding how each phase of a particular attack can be detected and thwarted.

Furthermore, certain attack patterns may yield outcomes that help enable other attack patterns, as in the example of Fig. 22. This shows interrelated attack patterns leading to exploitation of a database. Each main box is a CAPEC attack pattern, broken into attack phases. The CAPEC-170 pattern (web application fingerprinting) identifies details of the target database, which helps the attacker choose one of the three subsequent attacks leading to database compromise.

Another important resource for cybersecurity analysis is STIX™, a structured and extensible language for cyber threat intelligence. The unifying architecture of STIX encompasses cyber threat actors; their campaigns; their tactics, techniques, and procedures (TTPs); cyber incidents; attack indicators and observables; attack targets; and responsive courses of action. For example, Fig. 23 shows STIX content that describes a threat actor leveraging a CAPEC attack pattern (phishing) to deliver a particular kind of malware (The MITRE Corporation, 2016) as their TTPs. Another example is Fig. 24, in which STIX describes a particular indicator (intrusion detection rule) for data exfiltration. Particularly relevant to CyGraph models is STIX support for chaining of attacks (The MITRE Corporation, 2016).

For managing security information and events in their network environment, security teams often rely on SIEM products. These products collect data from a variety of sources (e.g., security devices, sensors, logs, and traffic) into a unified framework. ArcSight's Common Event Format (CEF) (ArcSight) is an interoperability standard for sharing such data. For example, it is possible to ingest STIX threat intelligence into ArcSight (via CEF) and correlate STIX fields with security events (Murdock and Pramanik, 2004).

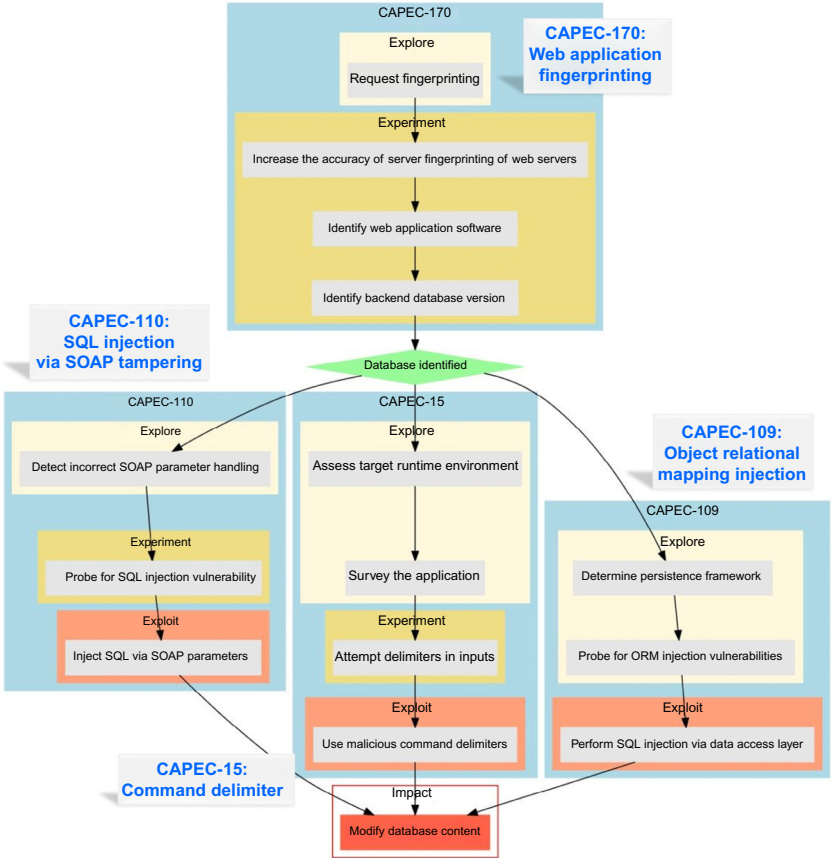

FIG. 22 Relationships within and between CAPEC attack patterns.

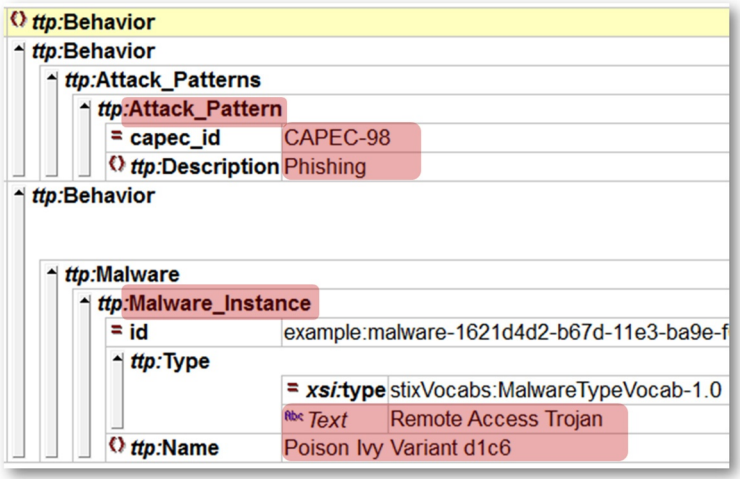

FIG. 23 STIX threat intelligence for attacker TTPs.

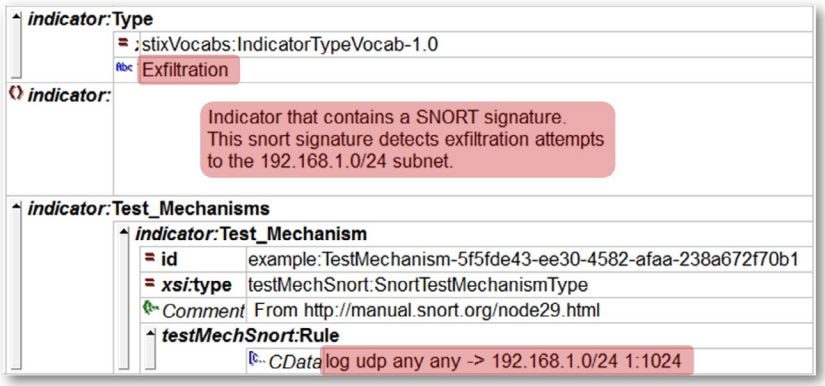

FIG. 24 STIX indicator for data exfiltration.

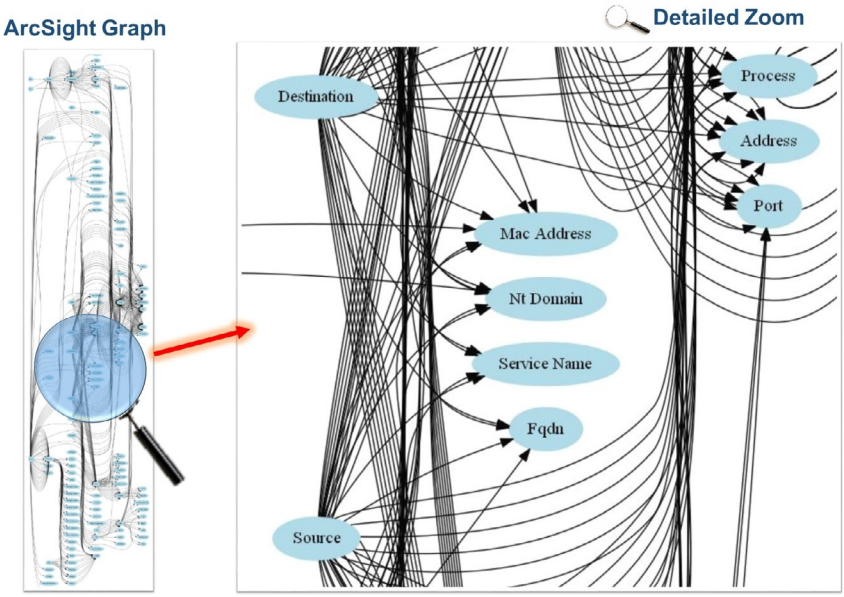

FIG. 25 Hierarchy of ArcSight field groupings.

We can leverage SIEMs for populating CyGraph models, for both cyber posture (e.g., vulnerabilities) and cyber threats (e.g., intrusion detection alerts). Fig. 25 shows the full range of ArcSight fields, which we cast as a directed acyclic graph for populating CyGraph. ArcSight organizes its data model as multilevel groupings of related fields. We capture this in graph relationships, where each edge denotes membership in a group (child as member of parent group).

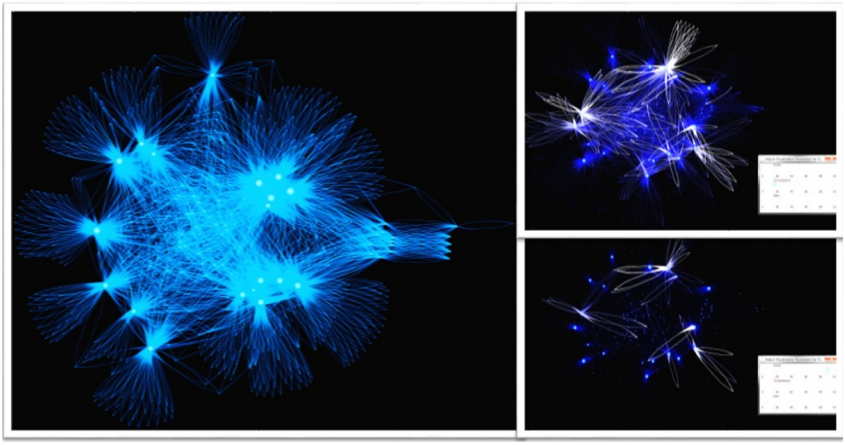

FIG. 26 Highlighting recent events in dynamically evolving graph model.

When the graph elements are labeled with time (network traffic, sensors, etc.), CyGraph can visualize graph evolution over time. When analyzing such time-dependent graph models, often it is important to maintain focus on recent events, so they do not become lost in the clutter of older ones. As shown in Fig. 26, CyGraph supports this through highlighting of recent events (edges). Here, network traffic is modeled with IP addresses as vertices, and edges for packets between them. More recent events (edges) are given brighter colors (nearly white), while older events are colored progressively darker (shades of blue (gray in the print version)) and more transparent.

4.1.3 Mission Dependencies

In CyGraph, the mission dependency layer analyzes the impact of cyberattacks on organizational functions (missions). It captures hierarchical dependencies among mission components, down to the cyber assets that support them. In practice, these dependencies are generally captured through manual modeling of mission dependencies, e.g., through CJA, CyCS, or CMIA. Fig. 27 is an example of such a mission dependency model ingested into CyGraph.

Through the mission dependencies layer of the knowledge graph, CyGraph shows transitive (*n*th order) mission effects of cyberattacks. For example, a CyGraph query can begin at the victim host of an attack and traverse the graph forward to enumerate the mission components that depend on it, showing impact on all effected levels of the mission dependency hierarchy. Such a query can also include potential next attack steps, following known vulnerability paths. A query could traverse in the opposite direction, to show the "cyber key terrain" supported by a given mission component. Overall, these kinds of analytic queries can tie together relationships among all layers of the CyGraph knowledge base.

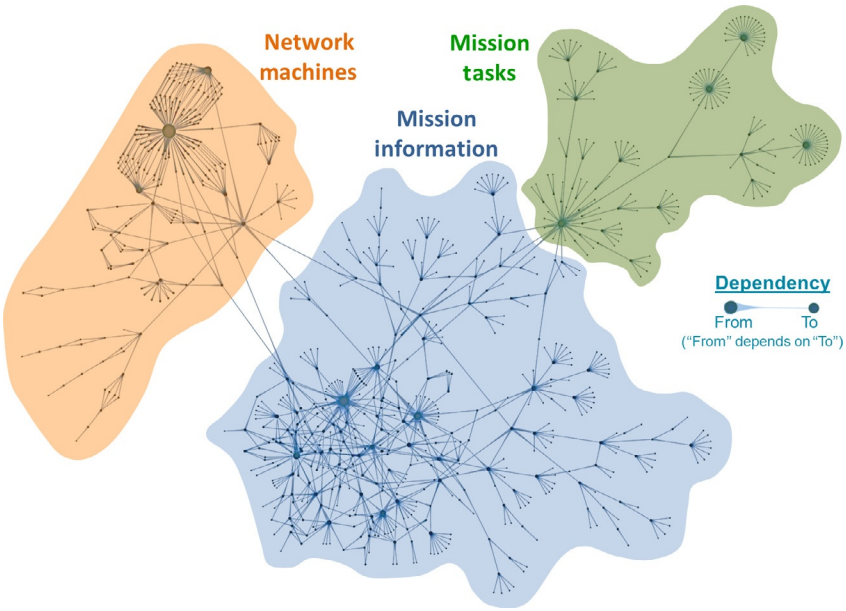

FIG. 27 Hierarchy of mission dependencies on cyber assets.

4.1.4 Case Study

This section presents a case study that illustrates various cybersecurity analytic capabilities in CyGraph. We compare a baseline model built by the TVA/Cauldron tool with a richer model built in CyGraph. CyGraph leverages the TVA/Cauldron output attack graph for the cyber posture layer of its knowledge graph. We then enrich this baseline model with intrusion detection alerts, vulnerability data from the NVD, and attack patterns from CAPEC™.

The left side of Fig. 28 shows the network architecture for this case study. In this architecture, the network is protected from the internet by an external firewall. Mission-critical servers are protected by a second (internal) firewall. There are workstations (represented by the vulnerable client machine in Fig. 28) that have two vulnerabilities—a remote buffer overflow in a web browser that allows arbitrary code execution, and stored credentials that can be stolen. The database backend server has an SQL injection vulnerability that is exploitable by an authenticated user. The Domain Name System (DNS) server in the network DMZ is vulnerable to a cache-poisoning attack that allows an attacker to redefine the internet address for a domain name, e.g., to an address under the attacker's control.

In this scenario, the vulnerable DNS service is exposed to the internet through the external firewall. The web client vulnerability is exposed to any malicious service to which the client browses, and the stored credential can be stolen by an attacker having sufficient control over this machine. The right side of Fig. 28 shows the resulting TVA/Cauldron attack graph, built from a

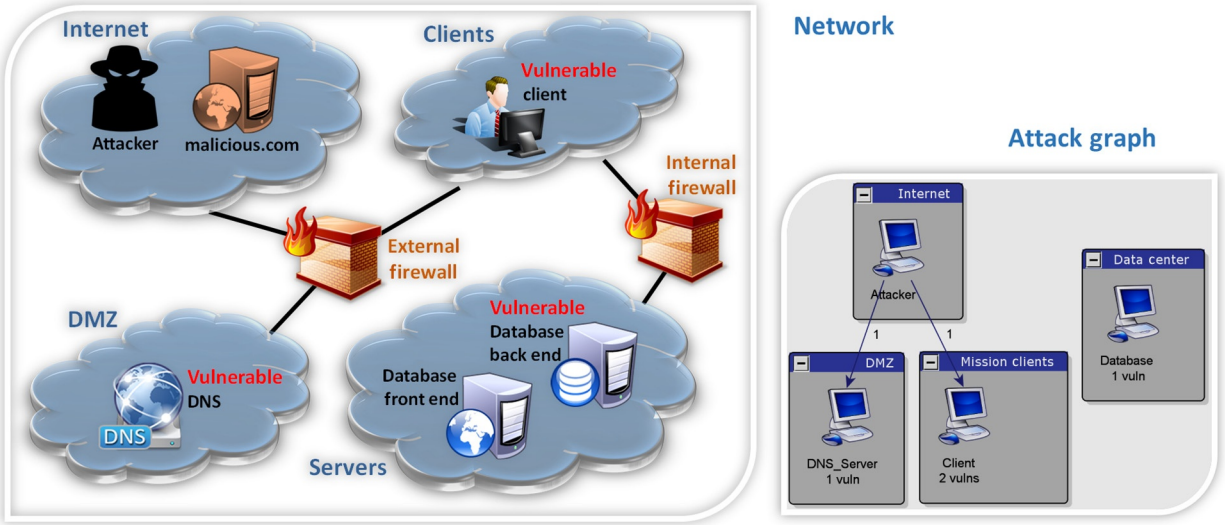

FIG. 28 Example network with TVA/Cauldron attack graph.

specification of the network topology, firewall rules, and vulnerability scan results. Of the four vulnerabilities reported in the scan, two are exposed to the internet, i.e., DNS server cache poisoning and web client remote buffer overflow. However, from its available information, the attack graph does not show any vulnerability exposures leading into the mission-critical servers.

Fig. 29 shows a CyGraph model for this scenario. It ingests the TVA/Cauldron attack graph, which gives machines, subnets (protection domains), and vulnerabilities (exposed through firewalls). We then add the network topology (connectivity among domains and firewalls), along with details about the vulnerabilities from the NVD and associated attack patterns from CAPEC. With the assumption that the servers are mission critical, this gives relationships in all four of the CyGraph knowledge graph layers.

This model of potentially exploitable attack paths provides context for responding to security events. When responding to events, queries anchored on events of interest allow the analyst to answer specific questions about the situation. In our scenario, an intrusion detection system (Snort) detects a buffer overflow attack against a network client machine (the cache-poisoning attack against the DNS server was not detected). To understand the context for this alert, the analyst submits this (Cypher) query:

```
MATCH paths = (:Machine)-[:SRC]->
   (:Alert {name:"Snort 33022"})-[:DETECTION]->
   (:Exploit)-[:AGAINST]->
   (:Vulnerability)-[:ON]->
   (:Machine)
RETURN paths
```

A literal translation of this query is “for this alert, show me the source (attacking) machine, and whether this alert is detection of exploitation against a vulnerability on a machine.”

Fig. 30 is the subgraph that matches this query. This shows that the victim of this alert does in fact have a vulnerability associated with an exploit (CAPEC attack pattern) that the intrusion signature (Snort 33022) detects. Here, there are additional relationships (not specified in the query) that Neo4j includes as associated with the returned nodes, i.e., DST for the alert’s destination machine and ENABLES representing the enabling of future possible exploitation.

Next, assume that a second alert is generated, which detects attempts at probing a web application for potential vulnerabilities. The analyst, already being suspicious about the first alarm (and the associated vulnerability), issues a query to analyze how the two alerts might be related. Here is the query:

```
MATCH paths = (:Alert {name:"Snort 33022"})-
   [:SRC|DST|DETECTION|ON|ENABLES|AGAINST|PREPARES*]->
   (:Alert {name:"Snort 1576"})
RETURN paths
```

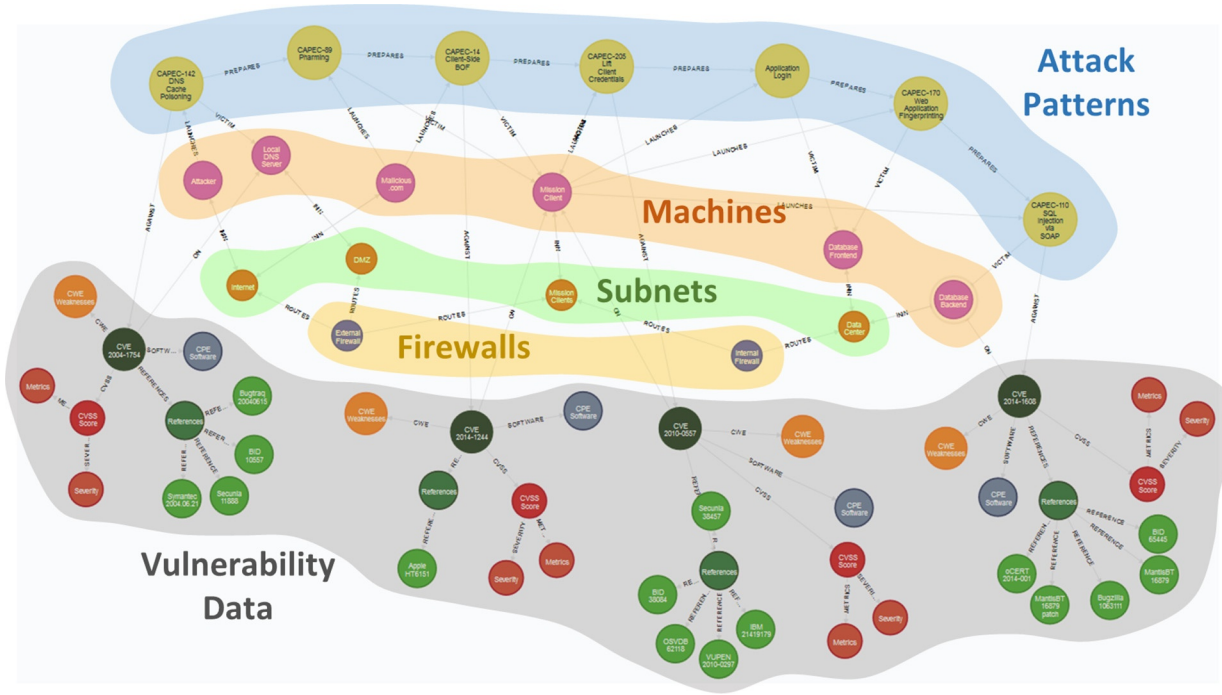

FIG. 29 Full cyber knowledge graph—ready for queries.

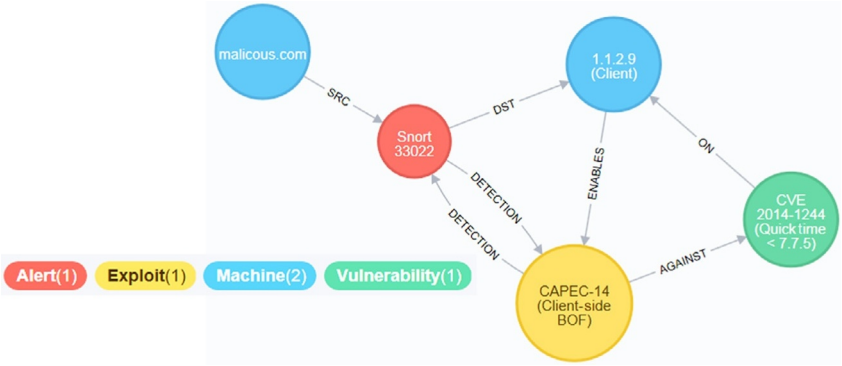

FIG. 30 Query showing context for an intrusion detection alert.

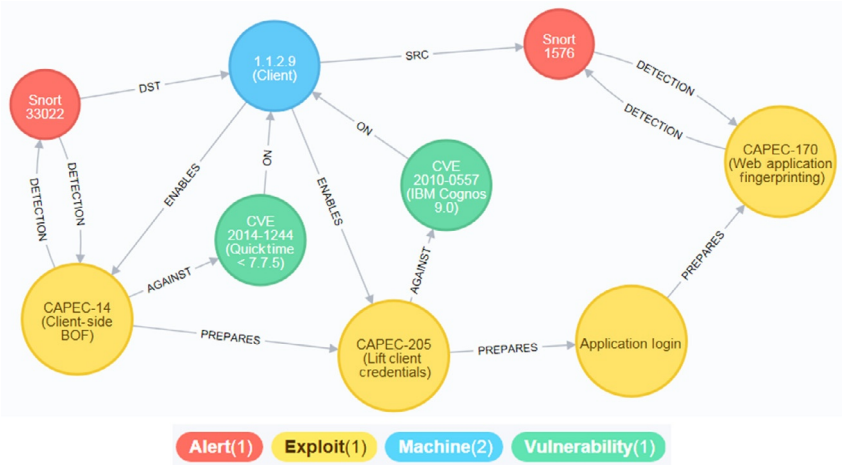

FIG. 31 Query showing relationships between two alerts.

This translates to "show me everything between these two alerts, using the relationship types about alerts and vulnerability exploitation." Here, the horizontal bars represent choice of relationship type, i.e., any of the given types will match. The asterisk at the end of the relationship types denotes traversal over an arbitrary number of relationships of these types, i.e., arbitrarily deep.

Fig. 31 is the resulting query match. This shows that the two alerts are indeed related, i.e., there is a chain of potential exploits linking them:

- Client-side buffer overflow against mission client.
- Lifting of database login credentials on client.
- Logging in to database from client.
- Fingerprinting to discover potential database vulnerabilities.

In this chain, two of the exploits are against known vulnerabilities. Just as for missed intrusion detections, vulnerability scanners do not always find existing vulnerabilities. In fact, in this case, the database login (after stealing the password) is essentially indistinguishable from a benign login and thus has no associated vulnerability or alert. Still, given this query result, the analyst might suspect that these are potentially multiple attack steps by the same threat actor.

Being concerned, the analyst might want to understand the next possible steps that the attacker could take. Here is a query to answer that:

```
MATCH paths = (:Alert {name:"Snort 1576"})-
   [:DST|DETECTION|ON|ENABLES|AGAINST|PREPARES*]->()
RETURN paths
```

This query translates to "show me everything that can happen after the second alert." Again, this uses a choice of relationship types about alerts and vulnerability exploitation, with arbitrarily deep traversal.

Fig. 32 is the subgraph match for this query. This shows that the web application fingerprinting against the database frontend server (which was detected) prepares for a subsequent attack against the database backend. Exploitation of this vulnerability would let the attacker inject arbitrary SQL commands, e.g., to steal, corrupt, or destroy mission-critical information.

At this point, the defender suspects that a malicious attack is being launched from the client. To better understand potential response options, the analyst poses this query:

```
MATCH
   ()-[r:ROUTES*]->(),
   (:Machine {name:"1.1.2.9 (Client)"})-[i1:IN]->(),
   (:Machine {name:"1.1.3.4 (Database Frontend)"})-[i2:IN]->(),
   (:Machine {name:"1.1.3.5 (Database Backend)"})-[i3:IN]->()
RETURN r, i1, i2, i3
```

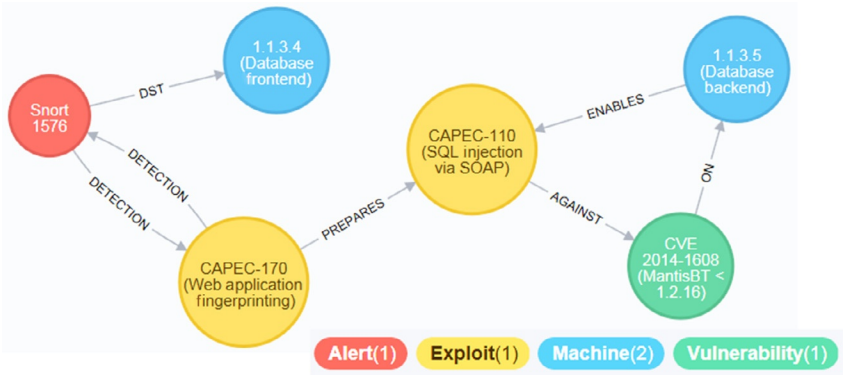

FIG. 32 Query showing potential mission impact.

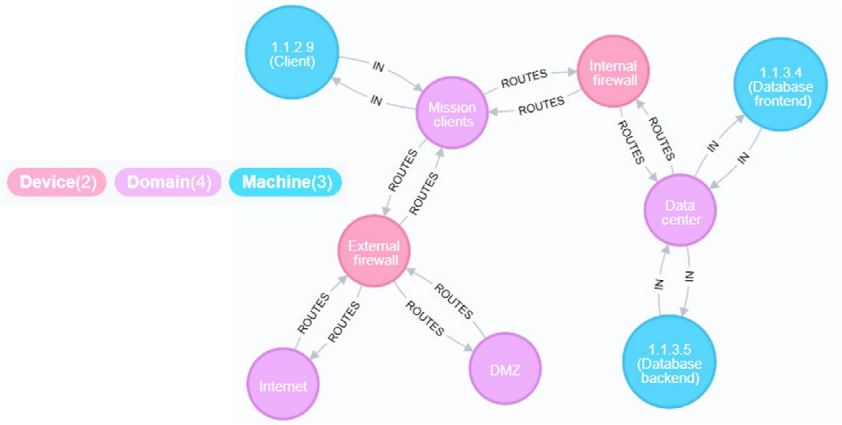

FIG. 33 Query showing potential firewall-blocking responses.

This translates to "show me the network topology, and how these three machines (client and two servers) connect to it."

Fig. 33 is the resulting query match. This shows that the internal firewall is in position to block traffic from the suspicious client to the mission-critical servers. Also, based on the correlation with the initial alert (client-side buffer overflow), the defender suspects that the network client is being controlled from the outside. For that, blocking via the external firewall is an option.

After the attack is thwarted, the defender wants to better understand how the attacker might have gained entry into the network. The defender poses this query:

```
MATCH paths = ()-[:PREPARES|ON|ENABLES|AGAINST*]->
   ()-[:PREPARES]->
   (:Exploit)-[:DETECTION]->
   (:Alert {name:"Snort 1576"})
RETURN paths
```

This translates to "show me all paths (of arbitrary depth) using relationship types about alerts and vulnerability exploitation (ignoring alert source/destination machines) that lead to an exploit detected by the initial alert."

Fig. 34 is the resulting query match. This suggests the cache-poisoning vulnerability as a likely precursor to the buffer overflow attack on the client, even though the cache-poisoning attack was not itself detected.

4.2 Cyber Modeling and Simulation

CyGraph can not only build a knowledge base from real network and threat data, but it can also be driven by simulations and perform model synthesis. Section 4.2.1 examines a CyGraph RESTful web service that visualizes

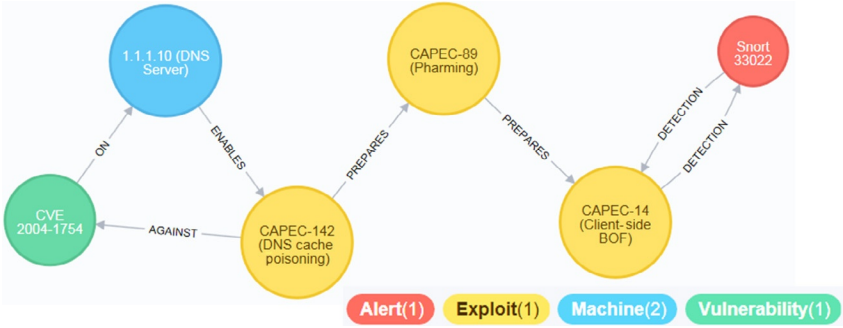

FIG. 34 Query discovering initial source of attack.

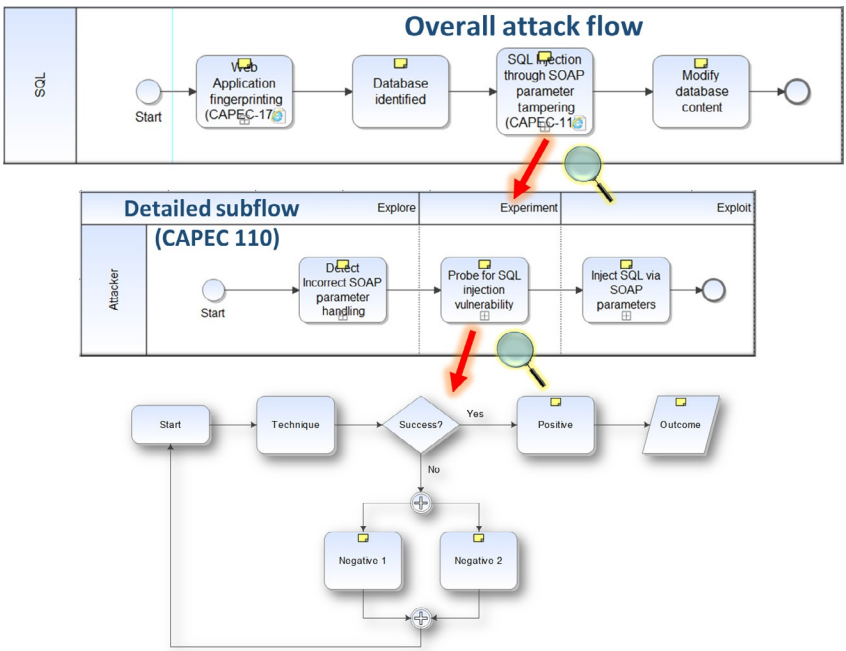

FIG. 35 Process flow model for simulation-driven analytics.

dynamic graph state changes. Section 4.2.2 applies CyGraph to the generation of node attributes for cyber graph models.

4.2.1 Simulation-Driven Visualizations

CyGraph includes a RESTful web service that visualizes dynamic state changes in graph models, e.g., driven by simulations. For example, Fig. 35 shows a process flow model for a multistep attack against a database, which

is modeled in terms of CAPEC attack patterns. This is modeled in iGrafx (Object Management Group, 2011), a tool that combines Business Process Model and Notation (BPMN) (iGrafx, 2016) and discrete event simulation. Such models are hierarchical, with high-level processes decomposed into lower-level process flows. In Fig. 35, the highest-level flow is a sequence of CAPEC attack pattern executions. Each attack pattern is decomposed into the explore/experiment/exploit phases defined by CAPEC, which are in turn decomposed into the detailed steps for carrying out each attack phase.

For sufficient realism, process models of cyberattacks need to be coupled with models of the network environment and security posture. For example, particular attack classes are only successful against certain classes of vulnerabilities, and firewall policies control attacker access. These kinds of vulnerability/attack conditions differ for every network and are continually evolving. They should be captured and managed by automated tools such as CyGraph.

This kind of hybrid modeling for cyber and mission interactions was explored in the AMICA project (Analyzing Mission Impacts of Cyber Actions) (Noel et al., 2015). AMICA captures behavior (process models) for mission operations and cyber (attacker and defender) activities, for high-fidelity understanding and measurement of mission impact. AMICA leverages CyGraph for knowledge management, automated model building, and visualization of environmental constraints (network topology, attack graph, mission dependencies, etc.) and dynamic state changes under simulation runs.

In AMICA, the iGrafx simulation engine follows the time and resource-constrained process model for mission and cyber threads. It tests cyber environmental constraints as needed in the process flows, updating them whenever process tasks (i.e., for cyber attacker and defender) change environmental conditions. Throughout the entire process, CyGraph shows the dynamic state evolution of the network environment through animated visualization.

Fig. 36 is an example of CyGraph visualization driven by simulations. This shows visualizations for two simulation runs, with different firewall policy rules for each run. In this scenario, firewall rules enforce access for an internal network and an external partner network. Policy 1 is coarse grained, allowing only mission-required access across the partner network domains, but within each domain (internal and partner) allowing fully connected access (even among machines that do not need to communicate for the mission). Policy 2 is more fine grained. The internal and partner networks are each divided into multiple domains, and only mission-required access provided across each of the smaller domains. In the scenario, the attacker starts at a particular host machine in the partner network and follows a shortest path to reach a critical database server.

In Fig. 36, the attack graph layer (left column) shows potentially exploitable paths of vulnerability from machine to machine. The network topology layer (right column) shows the underlying connectivity among hosts, switches, routers, and firewalls. The attack graphs have edges highlighted

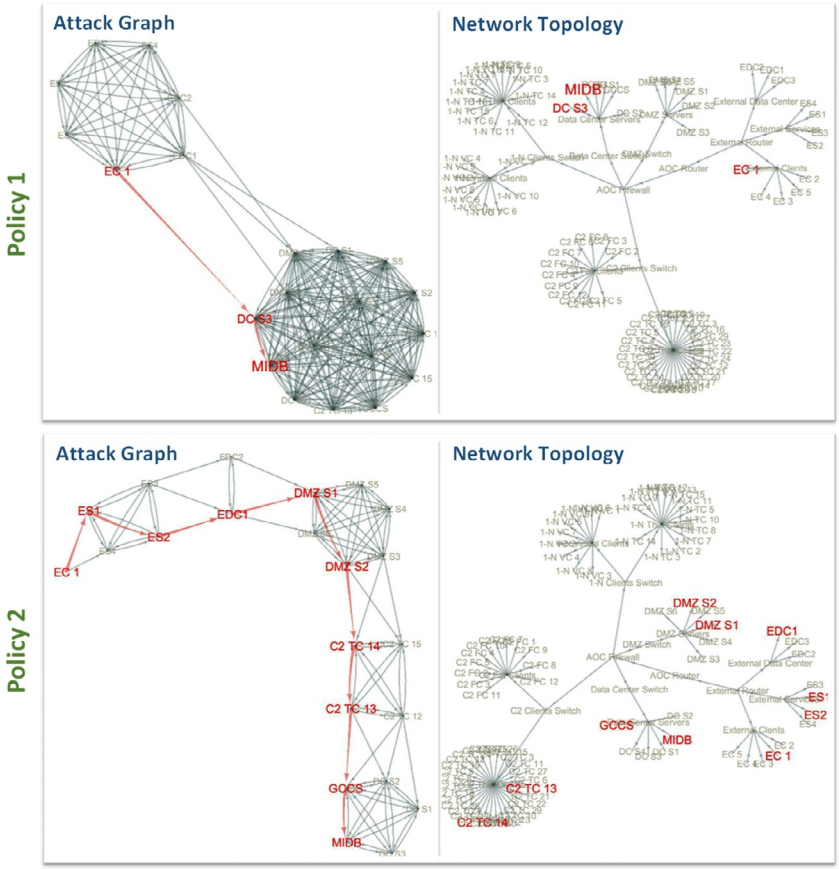

FIG. 36 CyGraph visualization driven by simulations.

representing paths of actual (simulated) exploitation, with the victim machines highlighted. The network topologies have these same victims highlighted. For the network topologies, each attack step (from one machine to another) involves multiple edges of the topology. Topology routes are generally shared over multiple attack steps, so that edge highlighting for multiple attack steps is not effective for topology graphs.

As shown in Fig. 36, the simplistic coarse granularity of Policy 1 allows the attacker to reach the target database service in only two attack steps. On the other hand, the more fine-grained (complex) and restrictive Policy 2 forces the adversary to attack nine hosts to compromise the target.

While not shown in this example, the state changes portrayed in CyGraph can be driven by both attacker and defender. For example, defender process flows can update the state of hosts and vulnerability exposures in response to attacks. In this way, the dynamic interplay of attackers and defenders under

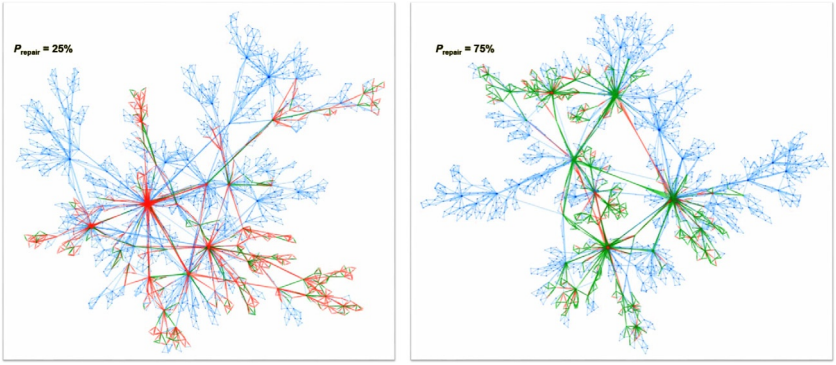

FIG. 37 Simulated interplay of cyber attacks and defenses.

simulation can be more deeply understood. Fig. 37 is an example of this. Here, the attacker is following a path of exploitation across the network. For each attack step, there is some probability that the attack is detected and thwarted by the defender. The figure shows simulation runs for two different defense-success probabilities, i.e., 25% and 75% (left and right sides of Fig. 37, respectively).

4.2.2 Cyber Model Synthesis

There are numerous practical reasons for the automatic generation of cyber graph models, e.g., for scalability testing or to avoid divulging sensitive real network data. There has been considerable progress in understanding the general laws and distinguishing characteristics of real-world graphs, for synthesizing graphs that are more realistic (Chakrabarti and Faloutsos, 2006). These generally focus on collective properties of graph nodes and edges, such as distribution of node degree, graph diameter, and community (clustering) structure. Thus, we can leverage established models for generating graphs (nodes and edges) themselves.

However, practical graph models also include various *attributes* for the nodes and edges in the model. For example, machines (nodes) in a network graph might include attributes such as numbers of vulnerabilities for each machine. We postulate that such attributes generally follow power-law distributions, in which the spectral density (data variance distributed over the frequency domain) varies inversely with frequency, known as $1/f$ models. Such distributions have been observed in many phenomenon (Wikipedia, 2016). One interpretation is a kind of preferential attachment in which some resource (e.g., vulnerabilities) are distributed among entities (e.g., machines) according to how many they already have—a kind of "like attracts like" argument (Merton, 1968). Another interpretation is a kind of stochastic noise process associated with a nonequilibrium system, subject to change driven by a flux of influences to/from other systems (Wikipedia, 2016).

In particular, we generate attribute values for nodes through pseudo-random noise, which has power-law distribution over a graph's spatial layout. This noise is two-dimensional (spatial), with a spectral density that varies inversely with spatial frequency. Such $1/f$ noise is also known as "pink noise" (Halley and Kunin, 1999), because of its color association with light spectra. It is intermediate between a purely random process having no memory of past values ("white noise") and random-walk processes that are dominated by their recent history ("brown noise").

We employ a typical approach for generating $1/f$ pink noise, which "reddens" white noise through a process of autoregression (smoothing). This is low-pass spatial filtering that reduces the higher-frequency components of the white noise (which itself contains equal amounts of all spatial frequencies). For white noise, the individual noise values are independent, i.e., uncorrelated over space. Pink noise varies more smoothly, so that nearby points are more correlated.

When we apply such $1/f$ (pink) noise distributions to generated graphs, nearby nodes are more likely to have similar attribute values, i.e., there is local correlation. For example, nearby nodes in a network structure might be more likely to be managed in a similar way, and thus have similar numbers of vulnerabilities. In rendering such discretized noise fields (two-dimensional arrays), we map noise values to display colors.

For generating pseudo-random $1/f$ spatial (two-dimensional) noise in CyGraph, we follow an established process (Vandevenne, 2004). We first create a two-dimensional array of random (white) noise values. We then apply linear interpolation as a smoothing process. This is done over multiple spatial scales (spatial frequencies), by successively varying the scale over which the smoothing is applied. This yields versions of the white noise smoothed at various scales, which we combine over all scales.

By controlling the relative contributions of smoothed noise at each scale, we obtain noise with different statistical distributions, as shown in Fig. 38. Noise signals with more smoothing (fewer high-frequency components) have stronger correlation among nearby values. For cyber graph models, this

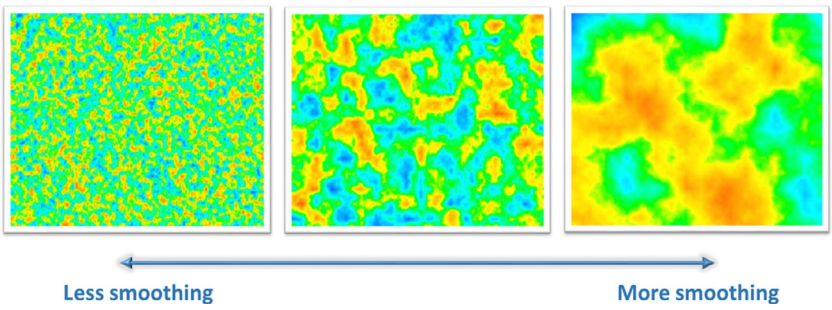

FIG. 38 Controlling variation in generated attribute values.

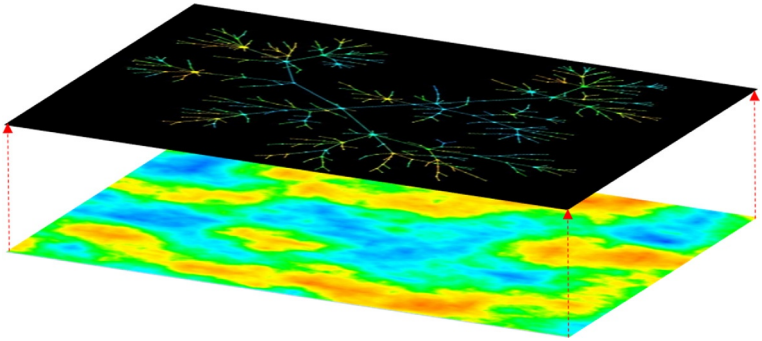

FIG. 39 Mapping correlated attribute values to graph nodes.

corresponds to environments with less variation in attribute values (e.g., vulnerabilities on machines).

Once we generate two-dimensional (spatial) 1/*f* noise with given statistical properties, we map the resulting noise array to a CyGraph model. In particular, we align the two-dimensional array of attribute values over a visualized graph, as shown in Fig. 39. The spatial coordinates of the graph nodes (e.g., network machines) are assigned attribute values (e.g., number of vulnerabilities) based on corresponding locations in the two-dimensional attribute (noise) array. The array values are not "noise" in the usual sense of unwanted contamination of a desired signal. Rather, they are the signal (node attribute) values themselves, as a model that is stochastic but still has locally correlated values (nearby nodes have similar attribute values).

In Fig. 39, the attribute values (two-dimensional 1/*f* noise array) are mapped to a color palette that helps distinguish noise values as bands of colors. The graph nodes are given the corresponding colors as projected from the noise array. The display also renders graph edges as the average of their endpoint node colors, to portray a visual transition.

5 SUMMARY

CyGraph is a flexible approach for dynamic cyber graph analytics and interactive visualization. It provides a graph knowledge base about attack vulnerability, threat indicators, and mission dependencies within a network environment. CyGraph builds a predictive model of possible attack paths and critical vulnerabilities and correlates network events to known vulnerability paths. It also includes dependencies among mission requirements and network assets, for analysis in the context of mission assurance. CyGraph has an open, extensible data model based on a layered property-graph formulation, which makes it easy to extend.

CyGraph brings together isolated data and events into an ongoing overall picture for decision support and situational awareness. It prioritizes exposed

vulnerabilities, mapped to potential threats, in the context of mission-critical assets. In the face of actual attacks, it correlates intrusion alerts to known vulnerability paths, and suggests best courses of action for responding to attacks. For postattack forensics, it shows vulnerable paths that may warrant deeper inspection.

CyGraph incorporates an attack graph model that maps the potential attack paths through a network. This includes any network attributes that potentially contribute to attack success, such as network topology, firewall rules, host configurations, and vulnerabilities. The dynamically evolving attack graph provides context for reacting appropriately to attacks and protecting mission-critical assets. CyGraph then ingests network events such as intrusion detection alerts and other sensor outputs, including packet capture. It also incorporates mission dependencies, showing how mission objectives, tasks, and information depend on cyber assets.

For building its graph knowledge base, CyGraph leverages a number of standards under Making Security Measurable™, a suite of standardization initiatives being developed by MITRE and others in the cybersecurity community. These provide standard languages for a variety of input data sources for building CyGraph knowledge graphs, particularly STIX™ (Structured Threat Information eXpression), CAPEC™, and content in the NVD.

In the CyGraph architecture, the data model is based on a flexible property-graph formulation. This model is free to evolve with the available data sources and desired analytics, rather than being fixed at design time. The backend database is implemented in Neo4j, a NoSQL database optimized for graphs. This represents node adjacency via direct pointers, avoiding expensive join operations for graph traversal. It allows CyGraph to take advantage of database technology with graph traversal performance orders of magnitude better than relational databases. Query execution times depend only on the size of the traversed subgraph, independent of the size of the overall graph.

In CyGraph, RESTful web services provide interfaces for data ingest, analytics, and visualization. Data in the wild are mapped to the common CyGraph data model. The domain-specific CyQL supports flexible ad hoc queries against the CyGraph data model. The middle-tier CyGraph Service then compiles CyQL into lower-level native graph query language (Cypher for Neo4j). CyGraph provides a variety of clients for specialized analytic and visual capabilities of query results, including dynamic graph evolution, layering, grouping/filtering, and hierarchical views.

CyGraph has the potential to greatly reduce effort (e.g., within enterprise security operations centers) for prevention and response of cyberattacks, provide situational awareness, and assure missions. Existing tools for attack graph analysis employ specialized data structures and algorithms designed for solving specific problems, e.g., attack reachability or network hardening. Flexibility and extensibility in the face of dynamically

evolving network environments and threats have not been first-class design criteria.

Overall, CyGraph introduces a novel unified data model that captures complex relationships among cyber security elements (network topology, firewalls, hosts, vulnerabilities, etc.), threat indicators (intrusion detection alerts, log file entries, suspicious traffic, etc.), and mission dependencies on cyber assets. This allows security operators to better understand the full scope of adversary activities, the relevance to known network vulnerability paths, and the potential impact to specific missions.

ACKNOWLEDGMENTS

This work was funded in part by the MITRE Innovation Program (project number EPF-14-00341), with Vipin Swarup as Cyber Security Innovation Area Leader. We wish to thank Bill Chan of MITRE for providing the architecture diagram for CAVE.

REFERENCES

Albanese, M., Jajodia, S., Noel, S., 2012. Time-efficient and cost-effective network hardening using attack graphs. In: Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks.

Amazon Web Services. AWS Management Portal for vCenter, 2016. https://aws.amazon.com/ec2/vcenter-portal/.

Andlinger, P., 2015. Graph DBMS Increased Their Popularity by 500% Within the Last 2 Years. http://db-engines.com/en/blog_post/43.

Angles, R., Gutierrez, C., 2008. Survey of graph database models. ACM Comput. Surv. 40 (1), 1–39.

ArcSight. ArcSight Publishes Open Standard Designed to Improve the Interoperability of Security and Compliance Systems. http://www.marketwired.com/press-release/arcsight-publishes-open-standard-designed-improve-interoperability-security-compliance-697462.htm.

Artz, M., 2002. NetSPA: A Network Security Planning Architecture. Master's thesis, Massachusetts Institute of Technology.

Baas, B., 2012. NoSQL spatial—Neo4j versus PostGIS. Master's thesis, Delft University of Technology.

Balzer, M., Deussen, O., 2005. Voronoi treemaps. In: Proceedings of the IEEE Symposium on Information Visualization.

Batra, S., Tyagi, C., 2012. Comparative analysis of relational and graph databases. Int. J. Softw. Eng. Soft Comput. 2, 509–512.

Bostock, M., Ogievetsky, V., Heer, J., 2011. D3: data-driven documents. IEEE Trans. Vis. Comput. Graph. 17 (12), 2301–2309.

Carrot Search. FoamTree: Interactive Voronoi Treemaps, 2016. http://carrotsearch.com/foamtree-overview.

Chakrabarti, D., Faloutsos, C., 2006. Graph mining: laws, generators, and algorithms. ACM Comput. Surv. 38, 1–69.

Cray. Urika-GD Product Brief, 2014. http://www.cray.com/sites/default/files/resources/Urika-GD%20Product%20Brief%20Online%205-page.pdf.

CyberAnalytix takes a 7-Year Path to $100 K, 2008. http://www.bizjournals.com/boston/blog/mass-high-tech/2008/05/cyberanalytix-takes-a-7-year-path-to-100k.html.

DB-Engines, 2015. DB-Engines Ranking—Trend of Graph DBMS Popularity. http://db-engines.com/en/ranking_trend/graph+dbms, March 2015.

Dutot, A., Guinand, F., Olivier, D., Pigné, Y., 2007. GraphStream: a tool for bridging the Gap between complex systems and dynamic graphs. In: Proceedings of the Emergent Properties in Natural and Artificial Complex Systems.

Graphviz—Graph Visualization Software. http://www.graphviz.org/.

Gudivada, V., Rao, D., Raghavan, V.V., 2016. Renaissance in database management: navigating the landscape of candidate systems. IEEE Comput. 49 (4), 31–42.

Gulabani, S., 2014. Developing RESTful Web Services with Jersey 2.0. Packt Publishing, Birmingham, UK.

Halley, J., Kunin, W., 1999. Extinction risk and the 1/f family of noise models. Theor. Popul. Biol. 56 (3), 215–230.

Harper, R., 2013. There Is Such a Thing as a Declarative Language, and It's the World's Best DSL. https://existentialtype.wordpress.com/2013/07/22/there-is-such-a-thing-as-a-declarative-language/.

Harris, E., Perlroth, N., 2014. Target missed signs of a data breach. The New York Times. http://www.nytimes.com/2014/03/14/business/target-missed-signs-of-a-data-breach.html.

Iannacone, M., Bohn, S., Nakamura, G., Gerth, J., Huffer, K., Bridges, R., Ferragut, E., Goodall, J., 2014. Developing an ontology for cyber security knowledge graphs. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference.

iGrafx, 2016. http://www.igrafx.com/.

Information Technology—Syntactic Metalanguage—Extended BNF, 1996. International Standard ISO/IEC 14977:1996(E).

Ingols, K., Lippmann, R., Piwowarski, K., 2006. Practical attack graph generation for network defense. In: Proceedings of the 22nd Annual Computer Security Applications Conference.

Jajodia, S., Noel, S., O'Berry, B., 2005. Topological analysis of network attack vulnerability. In: Kumar, V., Srivastava, J., Lazarevic, A. (Eds.), Managing Cyber Threats: Issues, Approaches and Challenges. Springer, Berlin, Germany.

Jajodia, S., Noel, S., Kalapa, P., Albanese, M., Williams, J., 2011. Cauldron: mission-centric cyber situational awareness with defense in depth. In: Proceedings of the 30th Military Communications Conference.

Lippmann, R., Ingols, K., 2005. An annotated review of past papers on attack graphs. Technical report, MIT Lincoln Laboratory.

Martin, R., 2008. Making security measurable and manageable. In: MILCOM 2008—2008 IEEE Military Communications Conference, 16–19 Nov., pp. 1–9, San Diego, CA.

Merton, R., 1968. The Matthew effect in science. Science 159 (3810), 56–63.

Murdock, J., Pramanik, S., 2004. Correlating Advanced Threat Information Feeds. HP Protect.

Musman, S., Tanner, M., Temin, A., Elsaesser, E., Loren, L., 2011. Computing the impact of cyber attacks on complex missions. In: Proceedings of the IEEE International Systems Conference.

Nadkarni, A., Vesset, D., 2014. Worldwide Big Data Technology and Services 2014–2018 Forecast. International Data Corporation. IDC #250458.

National Vulnerability Database, 2016. https://nvd.nist.gov/.

Neo4j, 2016. http://neo4j.com/.

Ning, P., Xu, D., 2004. Hypothesizing and reasoning about attacks missed by intrusion detection systems. ACM Trans. Inform. Syst. Secur. 7, 591–627.

Noel, S., 2015. Interactive visualization and text mining for the CAPEC cyber attack catalog. In: Proceedings of the ACM Intelligent User Interfaces Workshop on Visual Text Analytics.

Noel, S., Heinbockel, W., 2015. An overview of MITRE cyber situational awareness solutions. In: Proceedings of the NATO Cyber Defence Situational Awareness Solutions Conference.

Noel, S., Jajodia, S., 2007. Attack graphs for sensor placement, alert prioritization, and attack response. In: Proceedings of the Air Force Cyberspace Symposium.

Noel, S., Jajodia, J., 2009a. Attack Graph Aggregation. US Patent 7,627,900.

Noel, S., Jajodia, S., 2009b. Advanced vulnerability analysis and intrusion detection through predictive attack graphs. In: Proceedings of the Armed Forces Communications and Electronics Association Critical Issues in C4I.

Noel, S., Jajodia, S., 2014. Metrics suite for network attack graph analytics. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference.

Noel, S., O'Berry, B., Hutchinson, C., Jajodia, S., Keuthan, L., Nguyen, A., 2002. Combinatorial analysis of network security. In: Proceedings of the 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls.

Noel, S., Elder, M., Jajodia, S., Kalapa, P., O'Hare, S., Prole, K., 2009. Advances in topological vulnerability analysis. In: Proceedings of the Cybersecurity Applications & Technology Conference for Homeland Security.

Noel, S., Ludwig, J., Jain, P., Johnson, D., Thomas, R.K., McFarland, J., King, B., Webster, S., Tello, B., 2015. Analyzing mission impacts of cyber actions (AMICA). In: NATO Workshop on Cyber Attack Detection, Forensics and Attribution for Assessment of Mission Impact.

NSA-Funded 'Cauldron' Tool Goes Commercial, 2009. http://www.darkreading.com/nsa-funded-cauldron-tool-goes-commercial/d/d-id1131178.

O'Hare, S., Noel, S., Prole, K., 2008. A graph-theoretic visualization approach to network risk analysis. In: Proceedings of the Workshop on Visualization for Computer Security.

Object Management Group. Documents Associated with Business Process Model and Notation (BPMN) Version 2.0, 2011. http://www.omg.org/spec/BPMN/2.0/.

Ou, X., Govindavajhala, S., Appel, A., 2005. MulVAL: A Logic-Based Network Security Analyzer. In: Proceedings of the 14th USENIX Security Symposium.

Parr, T., 2013. The Definitive ANTLR 4 Reference. The Pragmatic Programmers.

RedSeal Networks, 2016. http://www.redsealnetworks.com/.

Ritchey, R., O'Berry, B., Noel, S., 2002. Representing TCP/IP connectivity for topological analysis of network security. In: Proceedings of the 18th Annual Computer Security Applications Conference.

Robinson, I., Webber, J., Eifrem, E., 2015. Graph Databases, second ed. O'Reilly, Sebastopol, CA.

Sanders, C., 2011. Practical Packet Analysis—Using Wireshark to Solve Real-World Problems, second ed. No Starch Press, San Francisco, CA.

Schweitzer, P., 2013. Attack–Defense Trees. Doctoral dissertation, University of Luxembourg.

Sheyner, O., Wing, J., 2004. Tools for generating and analyzing attack graphs. In: Proceedings of the Workshop on Formal Methods for Components and Objects.

Skybox. Risk Analytics for Cyber Security Management, 2016. http://www.skyboxsecurity.com/.

Structured Threat Information eXpression (STIX™)—A Structured Language for Cyber Threat Intelligence, 2016. https://stixproject.github.io/.

The MITRE Corporation. Making Security Measurable, 2013. http://makingsecuritymeasurable.mitre.org/.

The MITRE Corporation. Making Security Measurable—Directory of Efforts by Organization Name, 2014. http://makingsecuritymeasurable.mitre.org/directory/organizations/index.html.

The MITRE Corporation. The Common Attack Pattern Enumeration and Classification—A Community Resource for Identifying and Understanding Attacks, 2016. https://capec.mitre.org/.

The MITRE Corporation. Crown Jewels Analysis, 2009. http://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/crown-jewels-analysis.

The MITRE Corporation. Cyber Command System (CyCS), 2016. http://www.mitre.org/research/technology-transfer/technology-licensing/cyber-command-system-cycs.

The MITRE Corporation. Threat Actor Leveraging Attack Patterns and Malware, 2016. http://stixproject.github.io/documentation/idioms/leveraged-ttp/.

The MITRE Corporation. Kill Chains in STIX, 2016. http://stixproject.github.io/documentation/idioms/kill-chain/.

Vandevenne, L. Texture Generation using Random Noise, 2004. http://lodev.org/cgtutor/randomnoise.html.

vis.js—A Dynamic, Browser-Based Visualization Library, 2016. http://visjs.org/.

VMware. VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds, 2016. http://www.vmware.com/.

Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., Partner, J., 2015. Neo4j in Action. Manning Publications, Shelter Island, NY.

Wang, L., Yao, C., Singhal, A., Jajodia, S., 2006. Interactive analysis of attack graphs using relational queries. In: Damiani, E., Liu, P. (Eds.), Data and Applications Security XX. Lecture Notes in Computer Science. vol. 4127. Springer Berlin Heidelberg, Sophia Antipolis, France.

Wang, L., Jajodia, S., Singhal, A., Cheng, P., Noel, S., 2013. k-zero Day safety: a network security metric for measuring the risk of unknown vulnerabilities. IEEE Trans. Depend. Secure Comput. 11, 30–44.

Wikipedia. Power Law, 2016. https://en.wikipedia.org/wiki/Power_law.

Wikipedia. Non-Equilibrium Thermodynamics, 2016. https://en.wikipedia.org/wiki/Non-equilibrium_thermodynamics.

Wynn, J., Whitmore, J., Upton, G., Spriggs, L., McKinnon, D., McInnes, R., Graubart, R., Clausen, L., 2011. Threat assessment & remediation analysis (TARA): methodology description version 1.0. MITRE Technical report MTR110176.

Yates, C., Ladd, S., Devijver, S., 2006. Expert Spring MVC and Web Flow (Expert's Voice in Java). Apress, New York City.

Zadrozny, P., Kodali, R., 2013. Big Data Analytics Using Splunk. Apress, New York City.