

Big-Data Architecture for Cyber Attack Graphs

Representing Security Relationships in NoSQL Graph Databases

Steven Noel, Eric Harley, Kam Him Tam, and Greg Gyor

Cyber Security Division

The MITRE Corporation

McLean, Virginia, USA

[snoel, eharley, ktam, ggyor]@mitre.org

Abstract— The construction of cyber attack graphs is well studied in the literature. However, previous approaches generally apply only specific sets of algorithms against fixed data models. We introduce a much more flexible approach for attack graph modeling, analysis, and visualization based on general-purpose and extensible architecture. In this architecture, the model schema is implicit, and evolves with the available data sources, rather than being predetermined at design time. Graph pattern-matching queries provide flexible analytic capabilities against arbitrary graph structures. We then process query results in a generic way for visualization. Our highly scalable implementation includes the MongoDB document-oriented NoSQL database, Apache Spark for cluster computing, and the Neo4j graph database. We illustrate our approach with a novel attack graph model that merges a complex blend of network data, including topology, firewall policies, vulnerabilities, attack patterns, and intrusion alerts, through standardized languages for cyber security data.

Keywords— *attack graphs; topological vulnerability analysis; network attack modeling; cybersecurity standards; cluster computing; NoSQL databases*

I. INTRODUCTION

Network relationships are complex and can often be combined by adversaries in unexpected ways. We need to understand how cyber attackers can leverage initial footholds to extend their reach through the network. Rather than relying on perimeter defenses and being overwhelmed by many point sources, a unified picture of attacker movement through the network is needed.

We must also extend our notion of vulnerabilities, beyond the usual assumption of system flaws. Once inside a network, attackers can behave in ways that would otherwise be considered legitimate, while still achieving their goals. Furthermore, attacker movement does not depend on host vulnerabilities and configurations alone. The location of hosts within the overall network topology, security mechanism such as firewalls (both perimeter and internal), and specific access policy rules can all play roles.

Building such a comprehensive web of network attack relationships poses significant challenges. We must correlate data from numerous sources (topology, vulnerabilities, client/server configurations, policy rules, trust relationships, security events, etc.) into a common model. An architecture for

data representation and persistence must be flexible and easy to extend, and map naturally to network attack relationships. For interoperability with other tools, we should leverage security data standards where applicable.

Such a model has potential application to a variety of use cases (offensive/defensive, proactive/reactive, etc.), suggesting the need for ad hoc queries. In many cases, our query and analytic results are complex graph relationships, motivating the need for interactive graph visualization.

To address these challenges, we introduce an approach for mapping multi-step attack vulnerability and event dependencies through networks. We build a predictive model of possible attack paths, which we correlate with network events (intrusion alerts, sensor logs, etc.). We consider vulnerabilities not only in the sense of exploitable system flaws, but also legitimate accesses that can be leveraged once an attacker has established a foothold inside the network.

Correlating detected attack events with potential attack paths gives the best options for response, especially for protecting critical assets, and improves situational awareness, e.g., inferring missed attack steps and identifying false positives. We store the resulting attack graph in Neo4j [1], a NoSQL graph database. This is much more flexible for our problem domain than a relational database, and provides significantly better performance for graph analytics.

II. PREVIOUS WORK

A variety of approaches have been proposed for analyzing multi-step attacks through cyber networks. A predominant meta-model in this problem domain is to represent attack relationships as graphs or trees. The abstractions and semantics for these kinds of graphs vary depending on the network entities under consideration and the kinds of analytics supported.

For example, graphs can represent network state transitions leading to attack goals [2][3], attacker exploitation steps related by preconditions and postconditions [4][5], intrusion alert sequences [6][7], logical dependencies for attack goals [8][9], or host attack reachability [10][11][12]. Attack graphs have also been implemented with the relational database model [13]. We show results for the attack graph tool Cauldron [14] as a baseline of comparison for our much richer model.

A common theme in these kinds of approaches is to focus on a limited set of entity types in the graph model. These systems have generally been designed with fixed data models and algorithms, making them expensive to extend to new data sources and analytics. However, security concerns in real environments are complex and always evolving. Our aim is an extensible and scalable framework that can incorporate a rich variety of data about potential and actual adversary activities, supporting ad hoc queries and interactive visualization of query results.

Our data model incorporates standardized languages for cyber security (developed by MITRE and others) known collectively as Making Security Measurable™ [15]. This includes Common Vulnerabilities and Exposures (CVE)® [16], Common Vulnerability Scoring System (CVSS) [17], Common Weakness Enumeration (CWE)™ [18], Common Platform Enumeration (CPE)™ [19], and Common Attack Pattern Enumeration and Classification (CAPEC)™ [20].

III. APPROACH

Figure 1 shows the components of our system for mapping attack relationships through a network. We ingest data from various sources, map them to a common environment model. We analyze elements of the environment model for potential attack paths, and associate them with network events (alerts, logs, etc.). We store the resulting attack graph in a Neo4j database.



Figure 1. Components for cyber attack mapping.

We ingest data from various sources relevant to attacks (both potential and actual). For this, we rely primarily on the Splunk platform [21]. The ingest process maps the data to an agnostic (vendor-neutral) model using standardized language. The Cauldron tool ingests data from vulnerability scans and firewall configurations. We include additional sources for a much richer data model, including network flows, IDS/IPS alerts, anti-virus logs, operating system logs, host inventory agents, and proxy server logs. For mapping network attack relationships (e.g., testing source/destination/protocol/port combinations in firewall rules) we leverage Apache Spark [22], which has an in-memory compute model optimized for iterative computation on Apache Hadoop [23] clusters. As input, we build a model of the network environment and events, stored in MongoDB [24].

The result of our iterative analysis is a graph of potential attack steps and associated network events, which we store in a Neo4j graph database. Graph databases represent node adjacency without indices, via direct pointers. The relational model relies on expensive join operations (Cartesian products) when traversing graph edges, having complexity $O(n^d)$ for traversals of depth d over n graph nodes. In Neo4j, traversals follow direct links (edges only) so that complexity depends only on the size of query results sub-graph, independent of the total graph.

As a simple example, consider Figure 2. This illustrates the attack graph model employed by the Cauldron tool. It defines protection domains (e.g., a subnet or enclave), which contain machines with vulnerabilities. Implicitly, a machine can reach the vulnerabilities of all other machines in its domain. Graph edges across domains (exploits) are vulnerabilities that are exposed over the network (e.g., through firewalls) and reachable by machines in other domains.

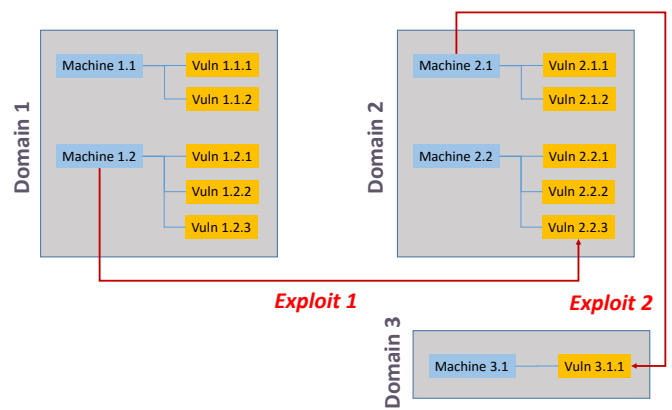
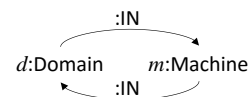


Figure 2. Protection domains, machines, vulnerabilities, and exploits.

Figure 3 shows our Neo4j representation of the attack graph in Figure 2. Neo4j employs the property-graph model [25]. A property graph contains nodes and nodes (vertices) and relationships (edges). Nodes and relationships are named; relationships are directed. Nodes and relationships can also contain properties, which are key-value pairs. A particular kind of property are labels, which define groups (types) of nodes or relationships.

Figure 3 has four node labels {Exploit, Machine, Vulnerability, Domain} and four relationship labels {IN, ON, LAUNCHES, AGAINST, VICTIM}. Machine-domain membership is



This represents that machine m is in protection domain d . The edges are needed in both directions to support graph traversal queries, which are directional. This allows traversal in both directions (from machine to domain and domain to machine), as needed to follow a particular flow of attack, including attack steps implicitly possible within a domain.

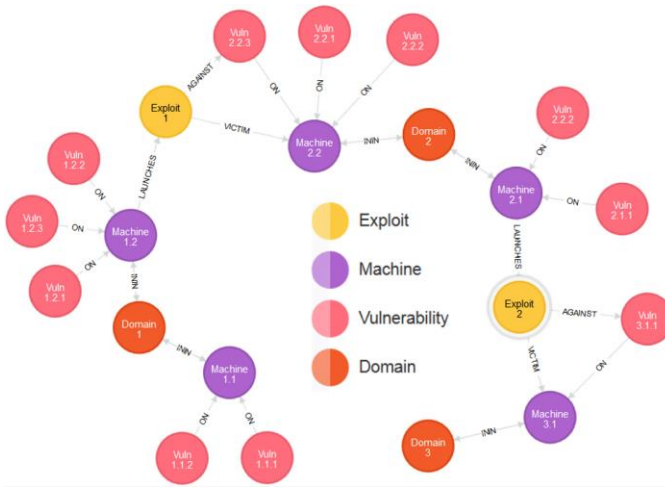


Figure 3. Attack graph represented as property graph.

This explicitly encodes the domain membership that is implicit in Cauldron. An advantage of the protection domain abstraction is that the number of edges among machines in a domain is linear in the number of machines rather than quadratic [26]. Our property-graph representation retains this advantage. However, in Cauldron, the implicit domain membership is built into the system, and cannot be changed for cases in which the assumption of full access among machines does not hold. In our flexible property-graph representation, we can simply omit protection domains, and explicitly connect machines through exploits.

Here is the representation for vulnerability v on machine m :

v :Vulnerability $\xrightarrow{:ON}$ m :Machine

An exploit e from machine m_1 against vulnerability v on machine m_2 is then

m_1 :Machine $\xrightarrow{:LAUNCHES}$ e :Exploit
 e :Exploit $\xrightarrow{:AGAINST}$ v :Vulnerability
 e :Exploit $\xrightarrow{:VICTIM}$ m_2 :Machine

With this basic attack graph representation as a property graph, we can perform the same kinds of analysis as the Cauldron tool. For this we leverage the Cypher query language [25] for Neo4j. For example, the Cypher query traverses the graph of Figure 3, finding all paths of exploitable vulnerabilities between a particular pair of machines:

```
MATCH path =
  (start:Machine {name:'Machine 1.1'})
  -[:LAUNCHES|VICTIM|IN*]->
  (end:Machine {name:'Machine 3.1'})
RETURN path
```

Figure 4 shows the result of this query.

This query seeks a sub-graph starting on Machine 1.1 (in Domain 1) and ending on Machine 3.1 (in Domain 3). It constrains the matching sub-graph to relationships of type LAUNCHES, VICTIM, or IN. This yields paths of Machine, Domain, and Exploit nodes. That is, it finds paths from the start machine to the end machine, through and across domains as required.

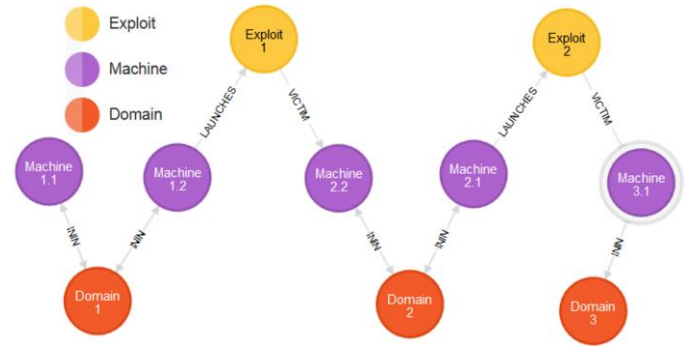


Figure 4. Query result for paths between two chosen machines.

IV. ILLUSTRATIVE EXAMPLE

In this section, we examine our approach to our cyber attack modeling. We first compare it to an existing attack graph tool (Cauldron). We then extend the baseline attack graph model with richer content for more in-depth analysis. Figure 5 shows the network for this illustrative example.

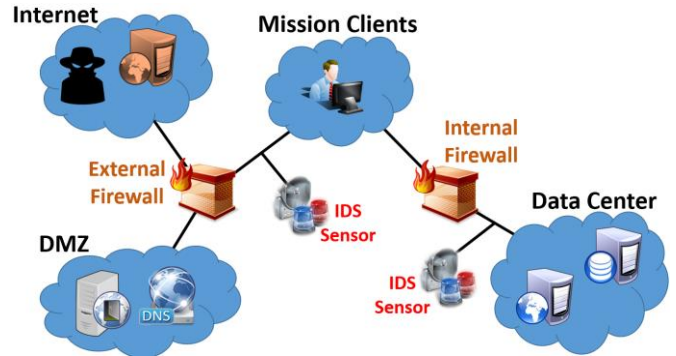


Figure 5. Network for illustrative example.

The internal network is segmented into three protection domains (DMZ, mission client workstations, and data center). The external firewall protects the internal network from the outside, and the internal firewall protects the critical data-center servers. Intrusion detection sensors monitor traffic crossing through the firewalls. Figure 6 is the resulting attack graph generated by the Cauldron tool.

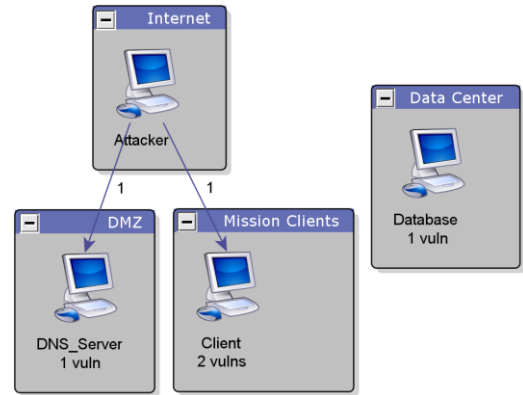


Figure 6. Cauldron attack graph for Figure 5.

This shows that the DNS server has a known vulnerability, which is exposed (through the external firewall) to the outside.

The client workstation has two vulnerabilities, one of which is exposed to the outside. In the data center, a database has a vulnerability, although it is blocked (by the internal firewall) from the other domains. While not shown here, Cauldron supports drilldown into the details for the vulnerabilities within and across domains. Assume those details are supplied by a vulnerability scanner in terms of CVE.

We build our model containing domains, machines, vulnerabilities, and exploits as described in the previous section. We then extend the model as additional nodes and relationships of our property graph. This includes the network topology (placement of firewalls between domains), attack patterns from CAPEC, vulnerability details from NVD, and intrusion alerts. Figure 7 is the resulting Neo4j property graph model for Figure 5.

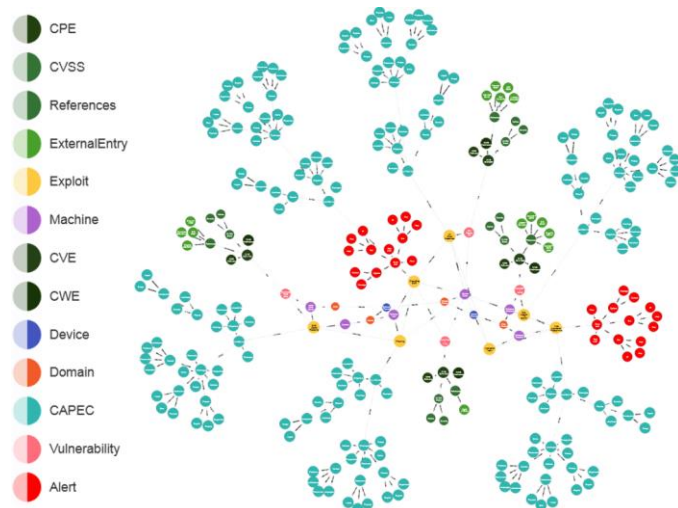


Figure 7. Neo4j property-graph model for Figure 5.

The property graph of Figure 7 is relatively complex. However, the intention is not to necessarily view the full graph. Rather, the property graph provides a rich framework for analytics and queries. For example, consider this query:

```
MATCH path = (start)-[r: IN|ROUTES*]->(end)
RETURN path
```

This query finds the IN and ROUTES relationships of the property graph, which define the network topology (domains, machines, and firewall devices). Figure 8 is the result of this query. This shows how firewalls separate domains, e.g., to identify the appropriate firewalls for network hardening.

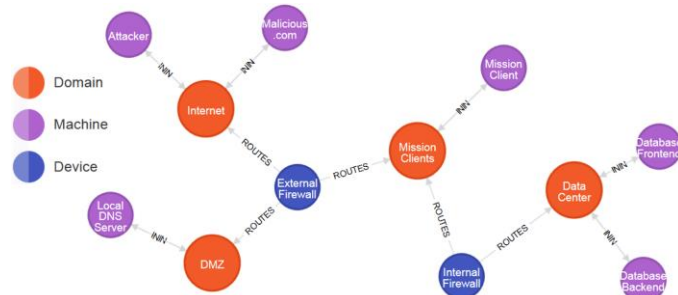


Figure 8. Query result showing network topology.

Next, assume that intrusion alerts are generated (red nodes in Figure 7). While Cauldron does not support such real-time events, this is a straightforward extension to our model. Consider then this query:

```
MATCH path = (start:Alert)
- [r:ALERT|AGAINST|VICTIM|ON*]->(end)
RETURN path
```

This query starts on alert nodes, and then traverses the ALERT, AGAINST, VICTIM, and ON relationships, resulting in Figure 9.

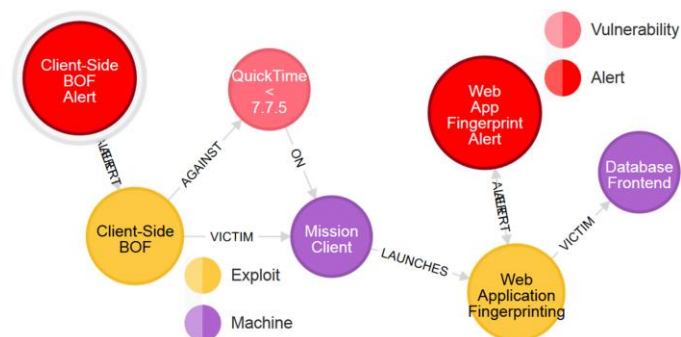


Figure 9. Vulnerable path between intrusion alerts.

By including the AGAINST, VICTIM, and ON relationships in our query, traversal leads away from alerts along paths of potential exploitation (via Exploit nodes). The LAUNCHES relationships is included because it spans the set of query result nodes. This query shows that the “client-side buffer overflow” alert (against a QuickTime vulnerability on the mission client) leads along an exploitable path to the “web application fingerprinting” alert (against the database front-end). From this, we might surmise that the alerts are potentially multiple attack steps by the same adversary.

We can expand the scope of exploits through this query:

```
MATCH path = (start:Alert)
- [r:ALERT|AGAINST|VICTIM|ON|LAUNCHES*]->(end)
RETURN path
```

This query adds the LAUNCHES relationship, to show all exploits launched when traversing away from alerts, not just immediate victims. The result is Figure 10.

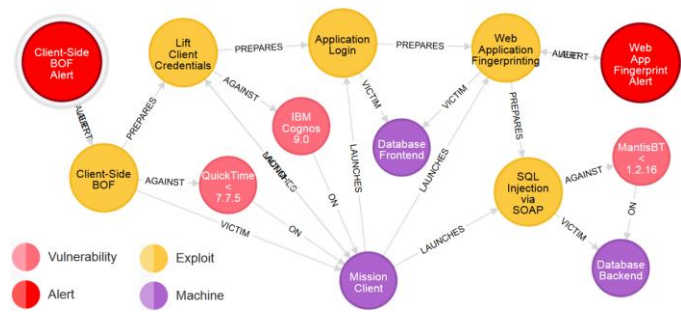


Figure 10. Vulnerable paths in the vicinity of alerts.

This shows that there is in fact a larger chain of potential exploits near the alerts:

- Client-side buffer overflow against mission client.
- Lifting of database login credentials on client.

- This ability to discover vulnerability paths between near alerts is especially important because in many cases key attacker behaviors go undetected.

```
MATCH path = (start)
    -[r:LAUNCHES|VICTIM|PREPARES*]->(end)
RETURN path
```

```

graph TD
    Malicious[Malicious.com] -- LAUNCHES --> Pharming[Pharming]
    Attacker[Attacker] -- LAUNCHES --> Pharming
    Pharming -- LAUNCHES --> ClientSide[Client-Side BOF]
    Pharming -- PREPARES --> ClientSide
    Pharming -- VICTIM --> MissionClient[Mission Client]
    ClientSide -- PREPARES --> MissionClient
    ClientSide -- VICTIM --> MissionClient
    LocalDNS[Local DNS Server] -- PREPARES --> Pharming
    LocalDNS -- VICTIM --> DNSCache[DNS Cache Poisoning]
    LocalDNS -- LAUNCHES --> SQLInjection[SQL Injection via SOAP]
    LocalDNS -- PREPARES --> SQLInjection
    SQLInjection -- LAUNCHES --> MissionClient
    SQLInjection -- VICTIM --> DatabaseBackend[Database Backend]
    Exploit[Exploit] -- PREPARES --> ApplicationLogin[Application Login]
    ApplicationLogin -- LAUNCHES --> MissionClient
    ApplicationLogin -- PREPARES --> DatabaseFrontend[Database Frontend]
    DatabaseFrontend -- VICTIM --> ApplicationLogin
    DatabaseFrontend -- VICTIM --> MissionClient
  
```

Figure 11. Chain of potential exploits with attackers and victims.

This query result introduces two preparatory exploits into the chain: cache poisoning by an outside attacker against the DNS server in the DMZ, followed by a pharming attack by a malicious web site. This leverages a vulnerability in the DNS server that allows an attacker to cause domain names to be resolved to a server under the attacker's control. The resulting pharming attack lets the malicious web site serve content that exploits the client-side vulnerability, compromising the client.

Our cyber-attack model leverages various standards defined under the Making Security Measurable framework. This includes CVE for known vulnerabilities, CVSS for vulnerability scoring and categorization, CWE for software weaknesses, and CPE for vulnerable software versions. Standardized language for security data helps close the gap for automated model generation. For example, many vulnerability scanners are CVE compliant, and the National Vulnerability Database (NVD) [27] maintained by NIST reports CVSS scores, CWE weaknesses, and CPE software versions for each CVE vulnerability.

For example, this query finds NVD content relevant to intrusion alerts:

```
MATCH path = (start:Alert)
- [r:ALERT | AGAINST | CVE | CVSS | SEVERITY |
  METRICS | CWE | CPE | REFERENCES |
  REFERENCE | CONTAINS* ] -> (end)
RETURN path
```

The diagram illustrates a network of relationships between various entities. The nodes are represented by colored circles, and the edges are labeled with terms indicating the nature of the relationships. The nodes and their connections are as follows:

- CVSS** (Green circle) is connected to **References** (Green circle) via an edge labeled **CVSS**.
- References** (Green circle) is connected to **ExternalEntry** (Green circle) via an edge labeled **REFERENCES**.
- ExternalEntry** (Green circle) is connected to **Exploit** (Yellow circle) via an edge labeled **EXTERNAL**.
- Exploit** (Yellow circle) is connected to **CVE** (Green circle) via an edge labeled **EXPLOIT**.
- CVE** (Green circle) is connected to **CWE** (Green circle) via an edge labeled **CVE**.
- CWE** (Green circle) is connected to **Vulnerability** (Pink circle) via an edge labeled **CWE**.
- Vulnerability** (Pink circle) is connected to **Alert** (Red circle) via an edge labeled **ALERT**.
- Alert** (Red circle) is connected to **Client-Side BOP Alert** (Red circle) via an edge labeled **ALERT**.
- Client-Side BOP Alert** (Red circle) is connected to **Client-Side BOF** (Yellow circle) via an edge labeled **ALERT**.
- Client-Side BOF** (Yellow circle) is connected to **QuickTime < 7.7.5** (Pink circle) via an edge labeled **AGAINST**.
- QuickTime < 7.7.5** (Pink circle) is connected to **CVE 2014-1244** (Green circle) via an edge labeled **CVE**.
- CVE 2014-1244** (Green circle) is connected to **CVSS Score** (Green circle) via an edge labeled **CVSS**.
- CVSS Score** (Green circle) is connected to **Metrics** (Green circle) via an edge labeled **METRICS**.
- Metrics** (Green circle) is connected to **Severity** (Green circle) via an edge labeled **SEVERITY**.
- CVE 2014-1244** (Green circle) is connected to **CWE Weaknesses** (Green circle) via an edge labeled **CWE**.
- CWE Weaknesses** (Green circle) is connected to **Severity** (Green circle) via an edge labeled **SEVERITY**.
- CVE 2014-1244** (Green circle) is connected to **References** (Green circle) via an edge labeled **REFERENCES**.
- References** (Green circle) is connected to **Apple HT6151** (Green circle) via an edge labeled **REFERENCE**.
- Apple HT6151** (Green circle) is connected to **Web App Fingerprint Alert** (Red circle) via an edge labeled **REFERENCE**.
- Web App Fingerprint Alert** (Red circle) is connected to **Web Application Fingerprinting** (Yellow circle) via an edge labeled **ALERT**.

Figure 12. Incorporating NVD content in property graph.

Our model also incorporates CAPEC, a community-developed catalog of common cyber attack patterns. CAPEC currently contains over 450 attack patterns, organized in a classification hierarchy of general to specific. CAPEC attack patterns capture knowledge about classes of attacks against cyber resources. Each attack pattern includes details about specific phases of the attack, the vulnerable attack surface, the resources required by the attacker, and ways to mitigate the attack. This provides a frame of reference for correlating various relevant information about attacks.

This query finds the CAPEC attack pattern content relevant to a particular intrusion alert:

```
MATCH path =(start:Alert
    {name:'Client-Side BOF Alert'})
    -[r: ALERT|AGAINST|CAPEC|PATTERN|
    CHALLENGE|PHASE|CONTEXT|
    MECHANISM|CONSEQUENCE|
    MITIGATION|IMPACT*]->(end)

RETURN path
```

Figure 13 is the query result.

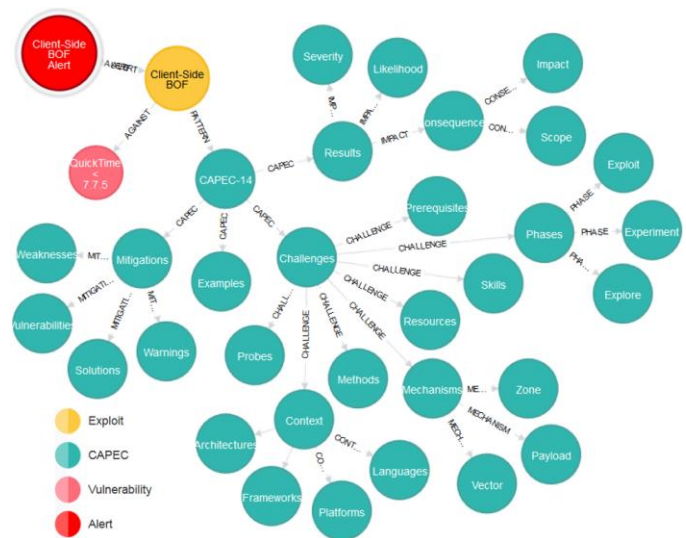


Figure 13. CAPEC attack pattern associated with an intrusion alert.

V. SUMMARY

We describe an open extensible modeling and analytical framework for tracing cyber-attack vulnerability paths through networks, correlated with observed security events. We consider a variety of relationships, such as network topology, firewalls, host vulnerabilities, potential attack patterns, and intrusion alerts. Our model leverages a property-graph formalism, implemented through Neo4j, a NoSQL graph database. Our modeling framework employs standardized data representations under the Making Security Measurable suite of languages.

We demonstrate how our approach supports the same kinds of analytics as an existing attack graph tool (Cauldron). We then extend the baseline analytics, developing a much richer model of the network environment and attacker activities. Through the application of the Neo4j Cypher query language, we iteratively explore attack relationships, building a more complete picture of security state.

Our work represents the first investigation of attack graph analysis based on NoSQL graph databases. These database architectures are optimized for graph analytics, especially traversals. This provides a flexible, scalable, high-performance persistence layer for our approach.

We provide the necessary context for responding to attacks, based on known vulnerability paths. We consider vulnerabilities not only in the usual sense of exploitable system flaws, but also legitimate accesses that can be leveraged once an attacker has established a foothold inside the network, e.g., capturing login credentials. In this way, we analyze system interdependencies to map how an attacker can incrementally penetrate a network. This makes our results particularly relevant to advanced persistent threats and malicious insider behaviors.

Our analysis guides the hardening of the network in advance of attacks. Potential attack paths are then correlated with detected attack events. We address the problem of missing alerts (false negatives), and extrapolate observed attacker presence to trace attack origins and next possible steps. This context gives the best options for response, especially for protecting critical assets, and improves situational awareness. Overall, we assemble disparate data sources into a global picture of overall security posture.

ACKNOWLEDGMENTS

This work was supported by the MITRE Innovation Program (project number EPF-14-00341), with Vipin Swarup as Information Area Lead.

REFERENCES

- [1] J. Partner, A. Vukotic, N. Watt, Neo4j in Action, Manning, 2014.
- [2] R. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," IEEE Symposium on Security and Privacy, 2000.
- [3] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," IEEE Symposium on Security and Privacy, 2002.
- [4] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," ACM Conference on Computer and Communications Security (CCS), 2002.
- [5] S. Jajodia, S. Noel, B. O'Berry, "Topological analysis of network attack vulnerability," Managing Cyber Threats: Issues, Approaches and Challenges, Springer, 2005.
- [6] P. Ning, Y. Cui, and D. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," ACM Conference on Computer and Communications Security (CCS), 2002.
- [7] S. Noel, E. Robertson, and S. Jajodia, "Correlating intrusion events and building attack scenarios through attack graph distances," Annual Computer Security Applications Conference (ACSAC), 2004.
- [8] S. Mauw, M. Oostdijk, "Foundations of attack trees," Lecture Notes in Computer Science Volume 3935, 2006.
- [9] S. Noel, S. Jajodia, L. Wang, A. Singhal, "Measuring security risk of networks using attack graphs," International Journal of Next-Generation Computing, 1(1), 2010.
- [10] K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, R. Cunningham, "Validating and restoring defense in depth using attack graphs," Military Communications Conference (MILCOM), 2006.
- [11] S. Jajodia, S. Noel, "Topological vulnerability analysis," Cyber Situational Awareness, Springer, 2010.
- [12] J. Hong and D. Kim, "Scalable security models for assessing effectiveness of moving target defenses," IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2014.
- [13] L. Wang, C. Yao, A. Singhal, S. Jajodia, "Implementing interactive analysis of attack graphs using relational databases," IFIP WG 11.3 Working Conference on Data and Applications Security, 2008.
- [14] S. Jajodia, S. Noel, P. Kalapa, M. Albanese, J. Williams, "Cauldron: mission-centric cyber situational awareness with defense in depth," Military Communications Conference (MILCOM), 2011.
- [15] R. Martin, "Making security measurable and manageable," CrossTalk: The Journal of Defense Software Engineering, September/October 2009.
- [16] The MITRE Corporation, Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names, <https://cve.mitre.org/>.
- [17] FIRST, Common Vulnerability Scoring System (CVSS-SIG), <http://www.first.org/cvss>.
- [18] The MITRE Corporation, Common Weakness Enumeration: A Community-Developed Dictionary of Software Weakness Types, <http://cwe.mitre.org/>.
- [19] National Institute of Standards and Technology, Official Common Platform Enumeration (CPE) Dictionary, <http://nvd.nist.gov/cpe.cfm>.
- [20] The MITRE Corporation, Common Attack Pattern Enumeration and Classification: A Community Resource for Identifying and Understanding Attacks, <https://capec.mitre.org/>.
- [21] D. Carasso, Exploring Splunk, CITO Research, 2012.
- [22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Fast and interactive analytics over Hadoop data with Spark," *login: Magazine*, 37(4), 2012.
- [23] M. Bhandarkar, "MapReduce programming with apache Hadoop," IEEE International Symposium on Parallel & Distributed Processing, 2010.
- [24] K. Chodorow, MongoDB: The Definitive Guide, O'Reilly Media, 2013.
- [25] I. Robinson, J. Webber, and E. Eifrem, Graph Databases, O'Reilly, 2013.
- [26] S. Noel, S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," ACM CCS Workshop on Visualization and Data Mining for Computer Security (VizSec), 2004.
- [27] National Institute of Standards and Technology, National Vulnerability Database: Automating Vulnerability Management, Security Measurement, and Compliance Checking, <http://nvd.nist.gov/>.