August 2020

Non-linear stochastic models for predicting exploitability

Sasith Maduranga Rajasooriya

Chris Peter Tsokos

Pubudu Kalpani K Hitigala Kaluarachchilage

US010754959B1

(12) **United States Patent**
Rajasooriya et al.

(10) **Patent No.: US 10,754,959 B1**
(45) **Date of Patent: Aug. 25, 2020**

(54) **NON-LINEAR STOCHASTIC MODELS FOR PREDICTING EXPLOITABILITY**

(71) Applicants: **Sasith Maduranga Rajasooriya**, Franklin, OH (US); **Chris Peter Tsokos**, Tampa, FL (US); **Pubudu Kalpani K Hitigala Kaluarachchilage**, Franklin, OH (US)

(72) Inventors: **Sasith Maduranga Rajasooriya**, Franklin, OH (US); **Chris Peter Tsokos**, Tampa, FL (US); **Pubudu Kalpani K Hitigala Kaluarachchilage**, Franklin, OH (US)

(73) Assignee: **University of South Florida**, Tampa, FL (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 200 days.

(21) Appl. No.: **15/875,249**

(22) Filed: **Jan. 19, 2018**

Related U.S. Application Data

(60) Provisional application No. 62/448,642, filed on Jan. 20, 2017.

(51) **Int. Cl.**

G06F 21/57	(2013.01)
G06N 7/00	(2006.01)
H04L 29/06	(2006.01)

(52) **U.S. Cl.**
CPC ***G06F 21/577*** (2013.01); ***G06N 7/005*** (2013.01); ***H04L 63/1433*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

10,268,660 B1 *	4/2019	Arazi	....	A63F 13/50
2006/0178887 A1 *	8/2006	Webber	....	G10L 15/144 704/256
2008/0010225 A1 *	1/2008	Gonsalves	....	G06N 7/005 706/11

(Continued)

OTHER PUBLICATIONS

Kaluarachchi, P.K., Tsokos, C.P. and Rajasooriya, S.M. (Apr. 2016) Cybersecurity: A Statistical Predictive Model for the Expected Path Length. Journal of information Security, vol. 7, 112-128. http://dx.doi.org/10.4236/jis.2016.73008.

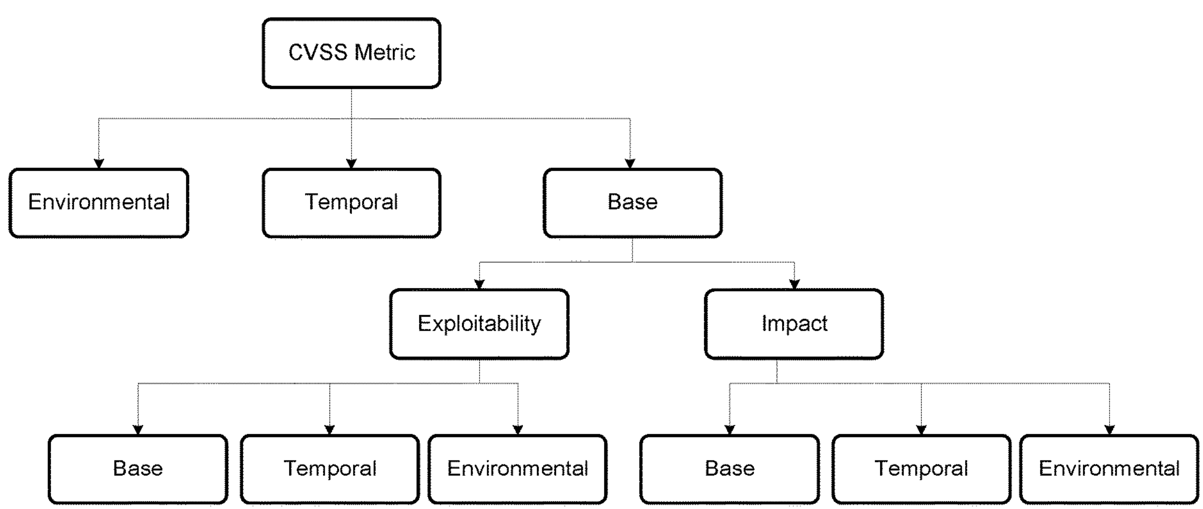
(Continued)

Primary Examiner — Kaveh Abrishamkar
(74) *Attorney, Agent, or Firm* — Thomas | Horstemeyer, LLP

(57) **ABSTRACT**

Procedures to identify the probabilities for different states in a vulnerability life cycle are described. The probabilities are used to develop a number of statistical models to evaluate the risk factor of a particular vulnerability at time “t”. A transition probability matrix of all states of a particular vulnerability as a function of time is also described. A Markov chain process can be iterated to reach a steady state of the transition probability matrix, with the initial probabilities reaching the absorbing states, including exploited and patched states. A risk factor is also introduced for use as an index of the risk of a vulnerability being exploited. Finally, statistical models that can calculate the risk factor more conveniently without going through the Markovian process are described.

18 Claims, 10 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

2016/0078309 A1*	3/2016	Feldman	G01B 9/0203 382/131	
2017/0046519 A1*	2/2017	Cam	G06F 21/577	

OTHER PUBLICATIONS

Sasith M. Rajasooriya, Chris P. Tsokos, Pubudu Kalpani Kaluarachchi (Jul. 2016) Stochastic Modelling of Vulnerability Life Cycle and Security Risk Evaluation. Journal of Information Security, vol. 7,269-279. http://dx.doi.org/10.4236/jis.2016.74022.
Rajasooriya, S.M., Tsokos, C.P. and Kaluarachchi, P.K. (Apr. 2017) Cyber Security: Nonlinear Stochastic Models for Predicting the Exploitability. Journal of Information Security, vol. 8, 125-140. http://dx.doi.org/10.4236/jis.2017.82009.
Kaluarachchi, P.K., Tsokos, C.P. and Rajasooriya, S.M. (Nov. 2017), Non-Homogeneous Stochastic Model for Cyber Security Predictions. Journal of Information Security,vol. 9, 12-24. https://doi.org/10.4236/jis.2018.91002.
Secunia Vulnerability Review 2015: Key Figures and Facts from a Global Information Security Perspective. Mar. 2015. https://secunia.com/?action=fetch&filename=secunia_vulnerability_review_2015_pdf.pdf.
NVD, National Vulnerability Database, Accessed Jun. 13 2018, http://nvd.nist.gov/.
Kijsanayothin, P. (May 2010) Network Security Modeling with Intelligent and Complexity Analysis. PhD Dissertation, Texas Tech University, Lubbock.
Alhazmi, O.H., Malaiya, Y.K. and Ray, I. (May 2007) Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems. Computers and Security Journal, vol. 26, Issue 3, pp. 219-228. doi:10.1016/j.cose.2006.10.002.
Noel, S., Jacobs, M., Kalapa, P. and Jajodia, S. (Nov. 2005) Multiple Coordinated Views for Network Attack Graphs. VIZSEC'05: Proc. of the IEEE Workshops on Visualization for Computer Security, Minneapolis, Oct. 2005, 99-106.
Mehta, V., Bartzis, C., Zhu, H., Clarke, E.M. and Wing, J.M. (Sep. 2006) Ranking Attack Graphs. In: Zamboni, D. and Krugel, C., Eds., Recent Advances in Intrusion Detection, RAID 2006, Lecture Notes in Computer Science, vol. 4219, Springer, Berlin, Heidelberg, 127-144. http://dx.doi.org/10.1007/11856214_7.
Frei, S. (2009) Security Econometrics: The Dynamics of (IN) Security, vol. 93, PhD Dissertation, ETH, Zurich.
Schiffman, M. (2014) Common Vulnerability Scoring System (CVSS). http://www.first.org/cvss/.
Bass, T. (Apr. 2000) Intrusion Detection System and Multi-Sensor Data Fusion. Communications of the ACM, vol. 43, Issue 4, pp. 99-105.
Jajodia, S. and Noel, S. (2005) Advanced Cyber Attack Modeling, Analysis, and Visualization. 14th USENIX Security Symposium, Technical Report, Mar. 2010, George Mason University, Fairfax, VA.
Abraham, S. and Nair, S. (Dec. 2014) Cyber Security Analytics: A Stochastic Model for Security Quantification Using Absorbing Markov Chains. Journal of Communications, vol. 9, No. 12, 899-907. https://doi.org/10.12720/jcm.9.12.899-907.
Wang, L., Singhal, A. and Jajodia, S. (Jul. 2007) Measuring Overall Security of Network Configurations Using Attack Graphs. Data and Applications Security XXI, Lecture Notes in Computer Science, vol. 4602. Springer, Berlin, Heidelberg , 98-112. https://doi.org/10.1007/978-3-540-73538-0_9.
Wang, L., Islam, T., Long, T., Singhal, A. and Jajodia, S. (Mar. 2008) An Attack Graph-Based Probabilistic Security Metric. DAS 2008, LNCS 5094, 283-296.
R statistics Tool. http://www.r-project.org, accessed Jun. 12, 2018.
Joh, H. and Malaiya, Y.K. (2010) A Framework for Software Security Risk Evaluation Using the Vulnerability Lifecycle and CVSS Metrics. Proceedings of the International Workshop on Risk and Trust in Extended Enterprises, Nov. 2010, 430-434.
Alhazmi, O.H. and Malaiya, Y.K. (Mar. 2008) Application of Vulnerability Discovery Models to Major Operating Systems. IEEE Transactions on Reliability, vol. 57, Issue 1, pp. 14-22. http://dx.doi.org/10.1109/TR.2008.916872.
Alhazmi, O.H. and Malaiya, Y.K. (Dec. 2005) Modeling the Vulnerability Discovery Process. Proceedings of 16th International Symposium on Software Reliability Engineering, Chicago, Nov. 8-11, 2005, 129-138. http://dx.doi.org/10.1109/ISSRE.2005.30.
CVE Details, Accessed on Jun. 14 2018, http://www.cvedetails.com/.
2016 U.S Government Cybersecurity Report Security Scorecard R&D Department, Apr. 2016. https://cdn2.hubspot.net/hubfs/533449/SecurityScorecard_2016_Govt_Cybersecurity_Report.pdf.
Symantec, Internet Security Threat Report 2016, vol. 21, Apr. 2016. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21/2016-en.pdf.
Dudorov, et al., Probability Analysis of Cyber Attack Paths against Business and Commercial Enterprise Systems, Intelligence and Security Informatics Conference (EISIC), 2013 European, Aug. 2013.
Marconato, et al., Security-related vulnerability life cycle analysis, 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS), Oct. 2012.
Marconato, et al., A Vulnerability Life Cycle-Based Security Modeling and Evaluation Approach, The Computer Journal, vol. 56, No. 4, Oxford University Press on behalf of the British Computer Society, Advanced Access on Sep. 3, 2012.
Mkpong-Ruffin, et al., Quantitative Software Security Risk Assessment Model, QoP '07 Proceedings of the 2007 ACM workshop on Quality of protection, Oct. 2007.
"H. Joh and Y. K. Malaiya, Defining and Assessing Quantitative Security RiskMeasures Using Vulnerability Lifecycle and CVSS Metrics, The 2011 International Conference on Security and Management (sam), Jul. 2011.".
Singhal, A. and Ou, X., Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs, NIST Interagency Report 7788, Aug. 2011, Computer Security Division Information Technology Laboratory National Institute of Standards and Technology, Gaithersburg, MD.
Xie, Lixia, Xiao Zhang, and Jiyong Zhang. "Network Security Risk Assessment Based on Attack Graph." Journal of Computers, vol. 8, No. 9 (Sep. 2013): 2339-2347.
Homer, John, Xinming Ou, and David Schmidt. "A sound and practical approach to quantifying security risk in enterprise networks." Kansas State University Technical Report, Computing and Information Sciences Department. Aug. 2009, pp. 1-15.
Secunia Research, "Key figures and facts on vulnerabilities from a global information security perspective", Vulnerability Review 2016, Mar. 2016, Flexera Software LLC.

* cited by examiner

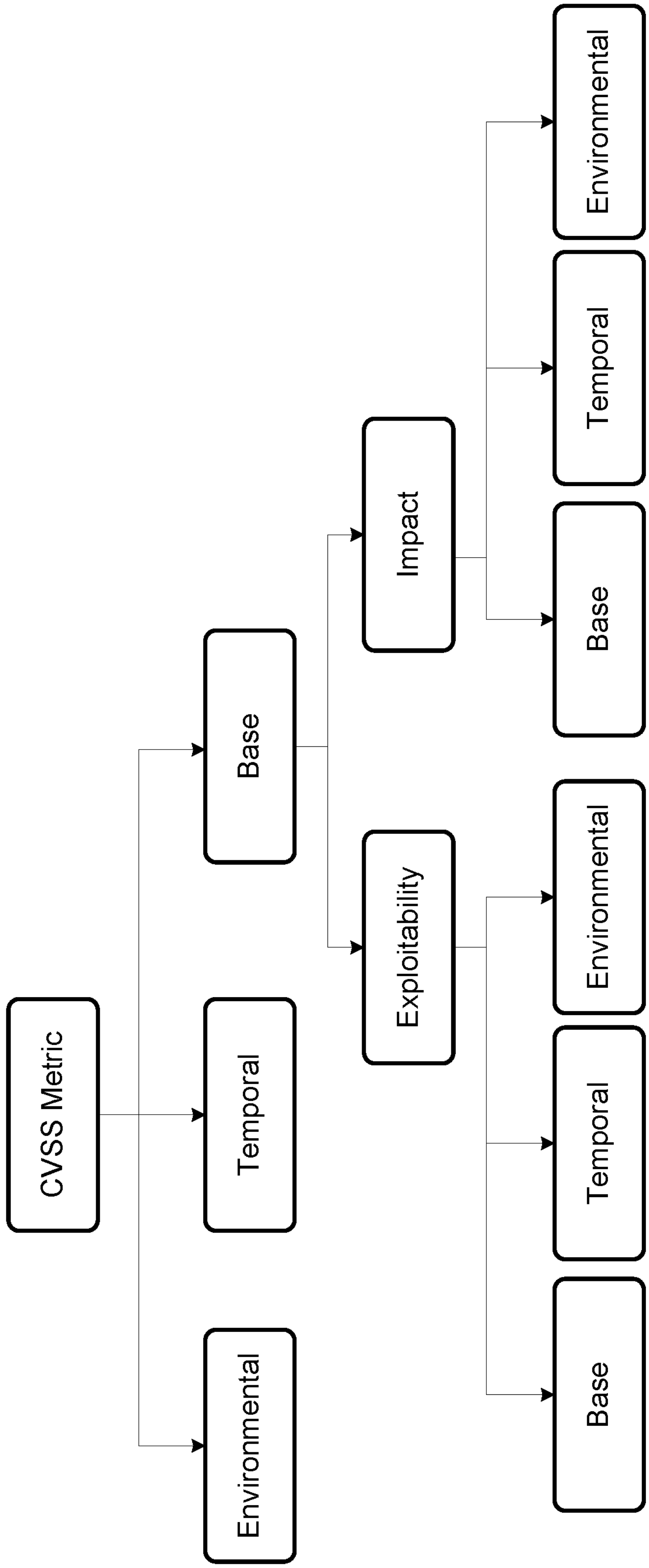

CVSS Metric
Base
Temporal
Environmental
Impact
Exploitability
Base
Temporal
Environmental
Base
Temporal
Environmental

FIG. 1

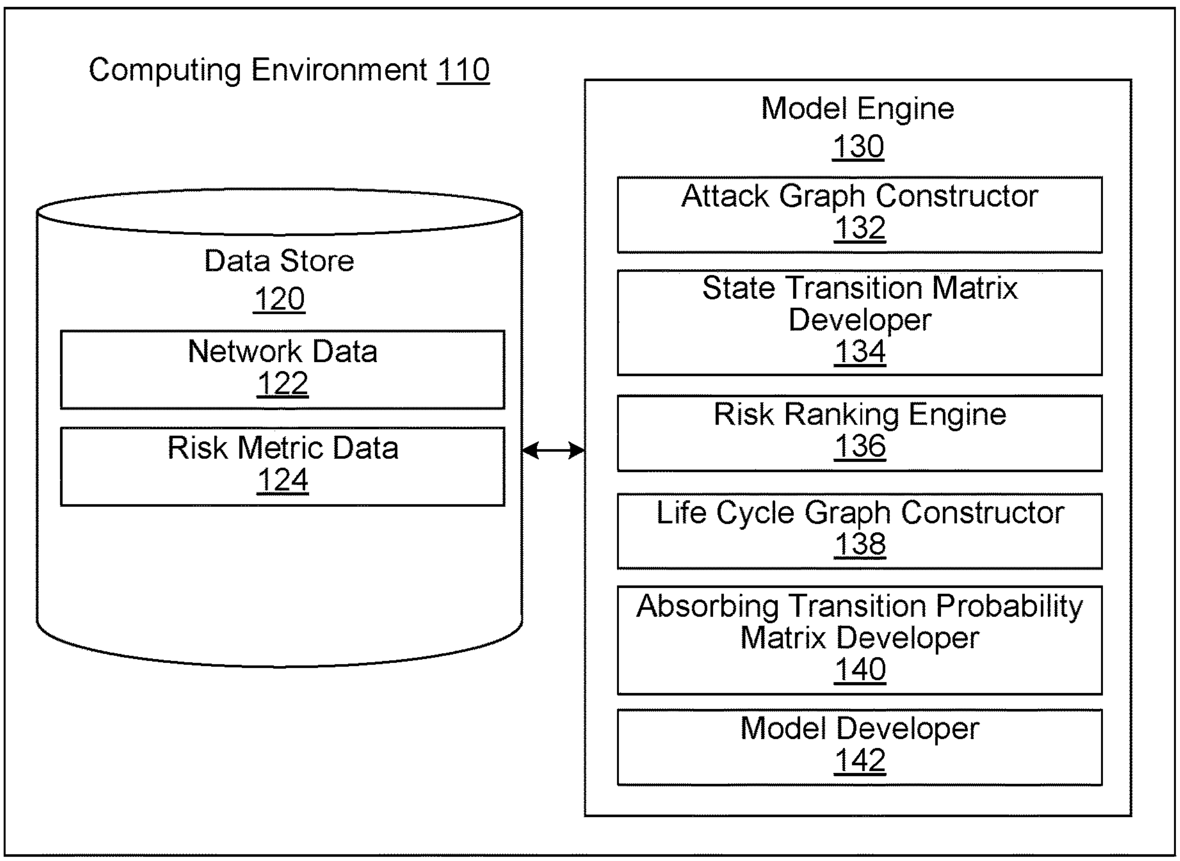

Computing Environment 110
Data Store
120
Network Data
122
Risk Metric Data
124
Model Engine
130
Attack Graph Constructor
132
State Transition Matrix Developer
134
Risk Ranking Engine
136
Life Cycle Graph Constructor
138
Absorbing Transition Probability Matrix Developer
140
Model Developer
142

FIG. 2

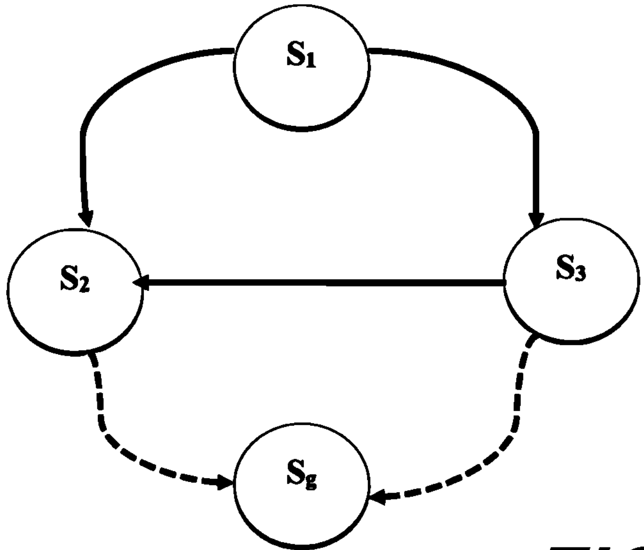


FIG. 3A

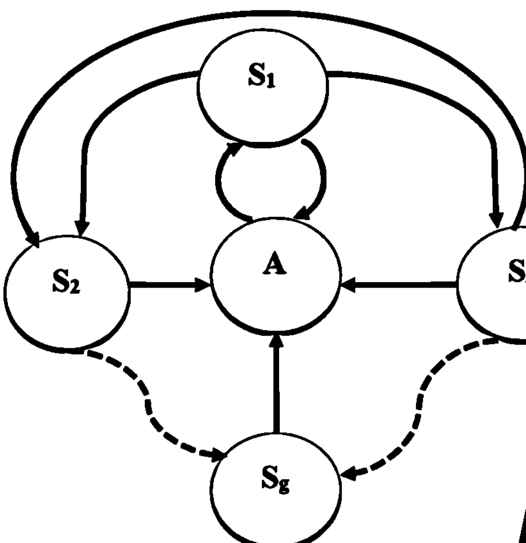


FIG. 3B

FIG. 4

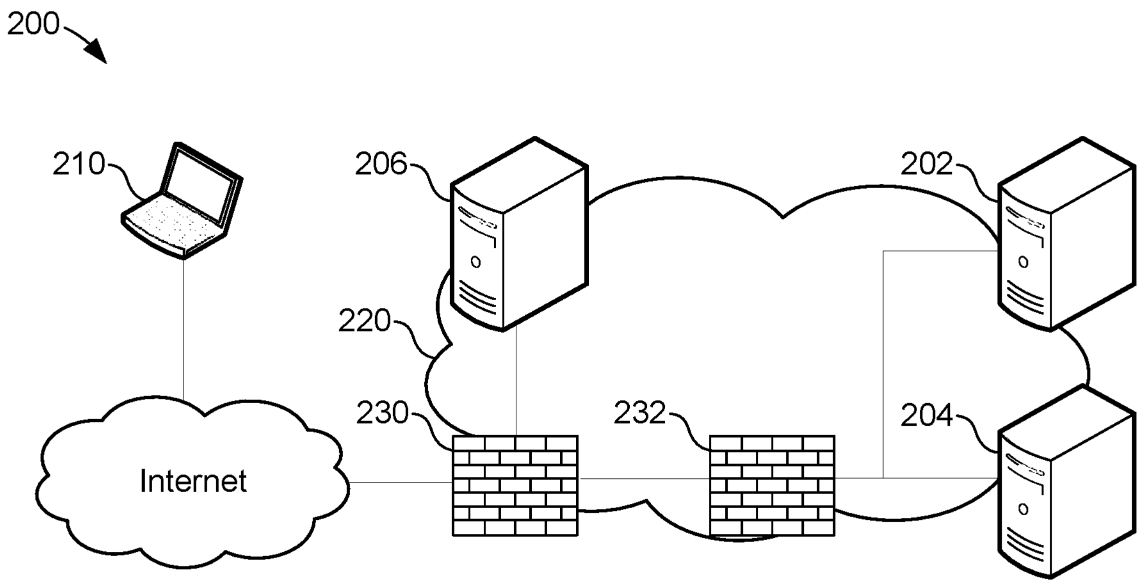

200
210
206
202
220
230
232
204
Internet

FIG. 5

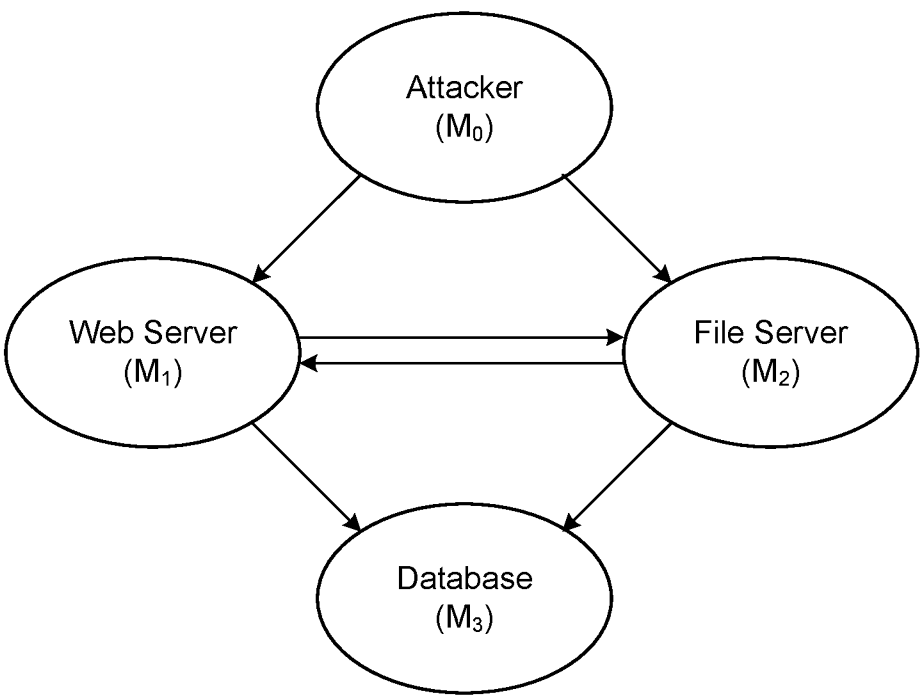

Attacker
(M_0)
Web Server
(M_1)
File Server
(M_2)
Database
(M_3)

FIG. 6

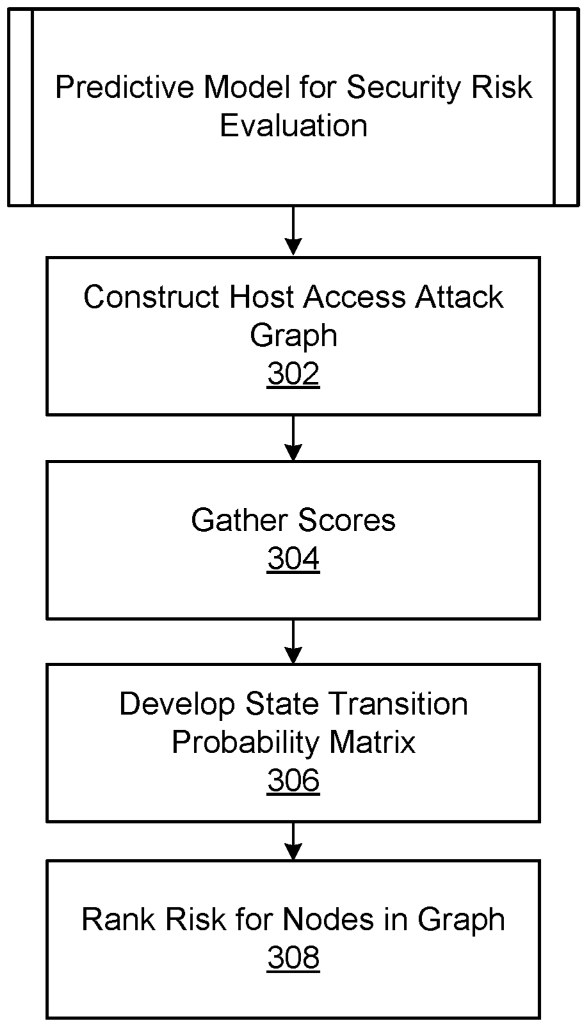

Predictive Model for Security Risk Evaluation
Construct Host Access Attack Graph
302
Gather Scores
304
Develop State Transition Probability Matrix
306
Rank Risk for Nodes in Graph
308

FIG. 7A

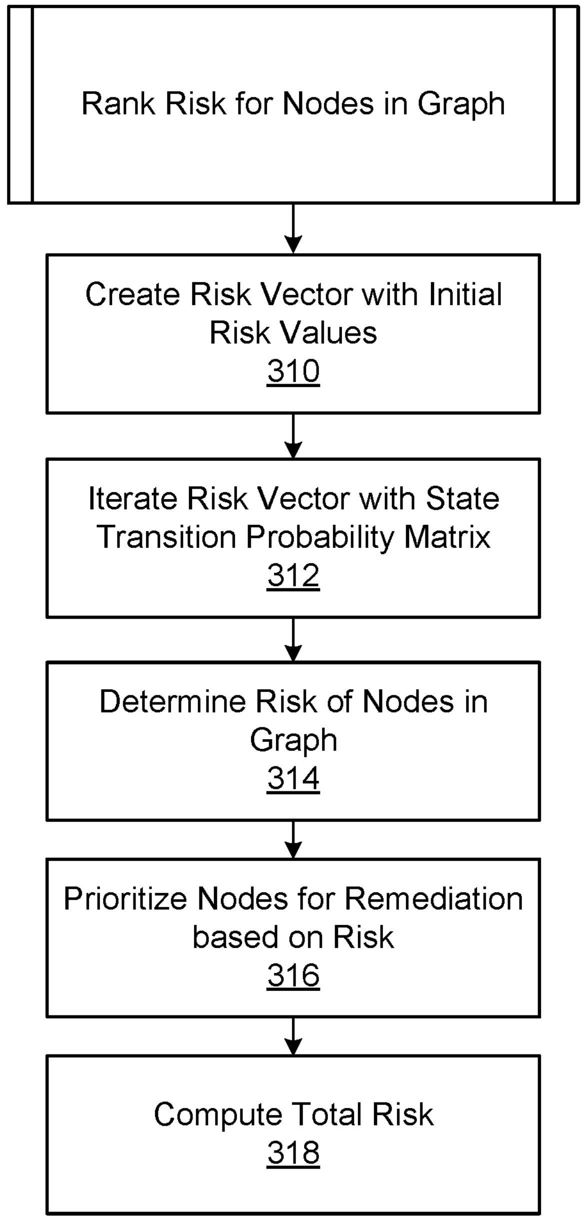

Rank Risk for Nodes in Graph
Create Risk Vector with Initial Risk Values
310
Iterate Risk Vector with State Transition Probability Matrix
312
Determine Risk of Nodes in Graph
314
Prioritize Nodes for Remediation based on Risk
316
Compute Total Risk
318

FIG. 7B

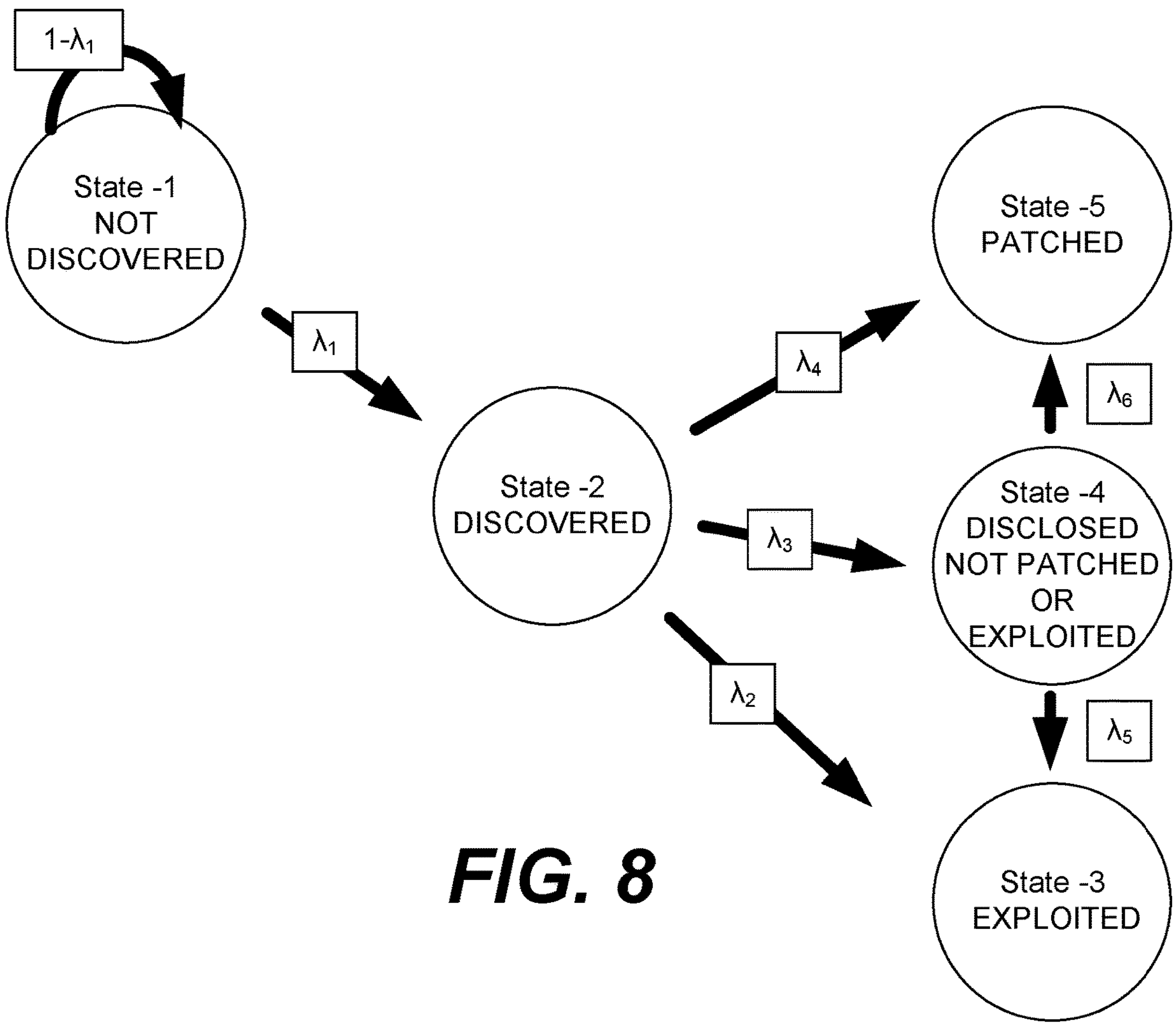

1-λ1
State -1
NOT
DISCOVERED
λ1
State -2
DISCOVERED
λ4
λ3
λ2
State -5
PATCHED
λ6
State -4
DISCLOSED
NOT PATCHED
OR
EXPLOITED
λ5
State -3
EXPLOITED

FIG. 8

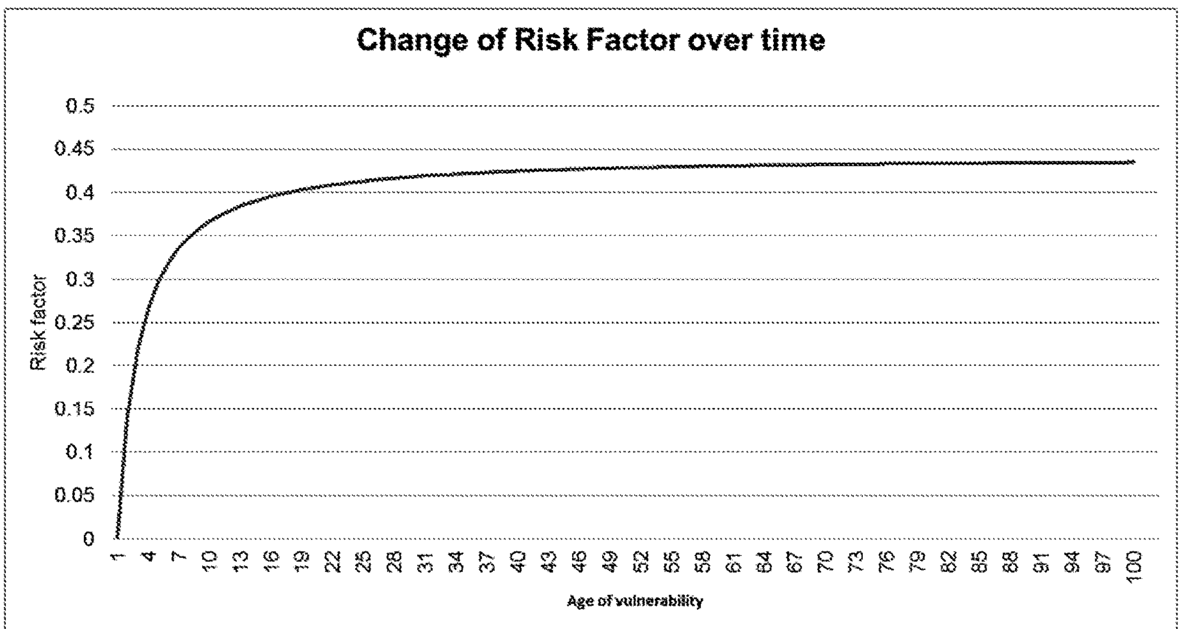

Change of Risk Factor over time
Risk factor
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
Age of vulnerability

FIG. 9

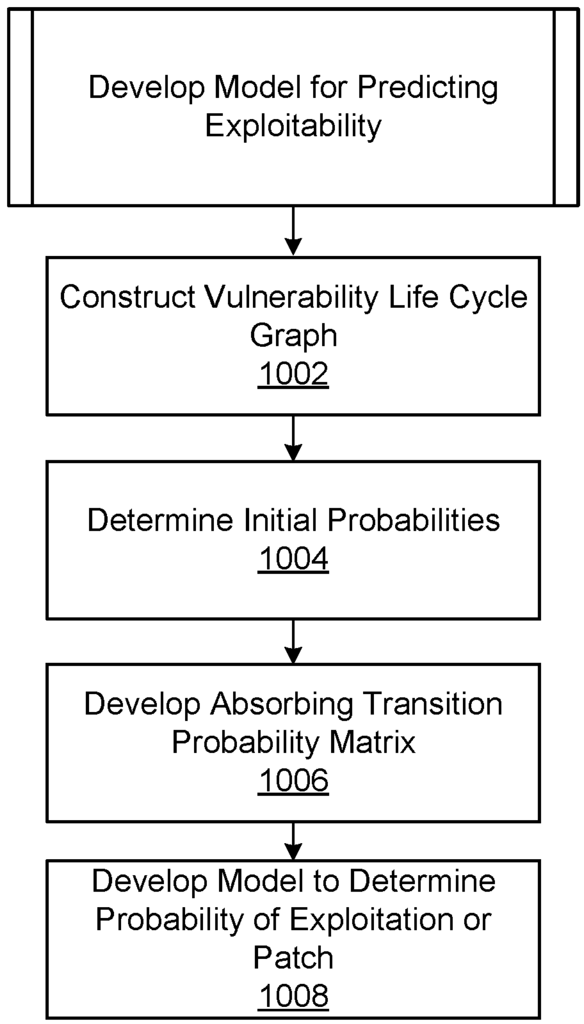

Develop Model for Predicting Exploitability
Construct Vulnerability Life Cycle Graph
1002
Determine Initial Probabilities
1004
Develop Absorbing Transition Probability Matrix
1006
Develop Model to Determine Probability of Exploitation or Patch
1008

FIG. 10

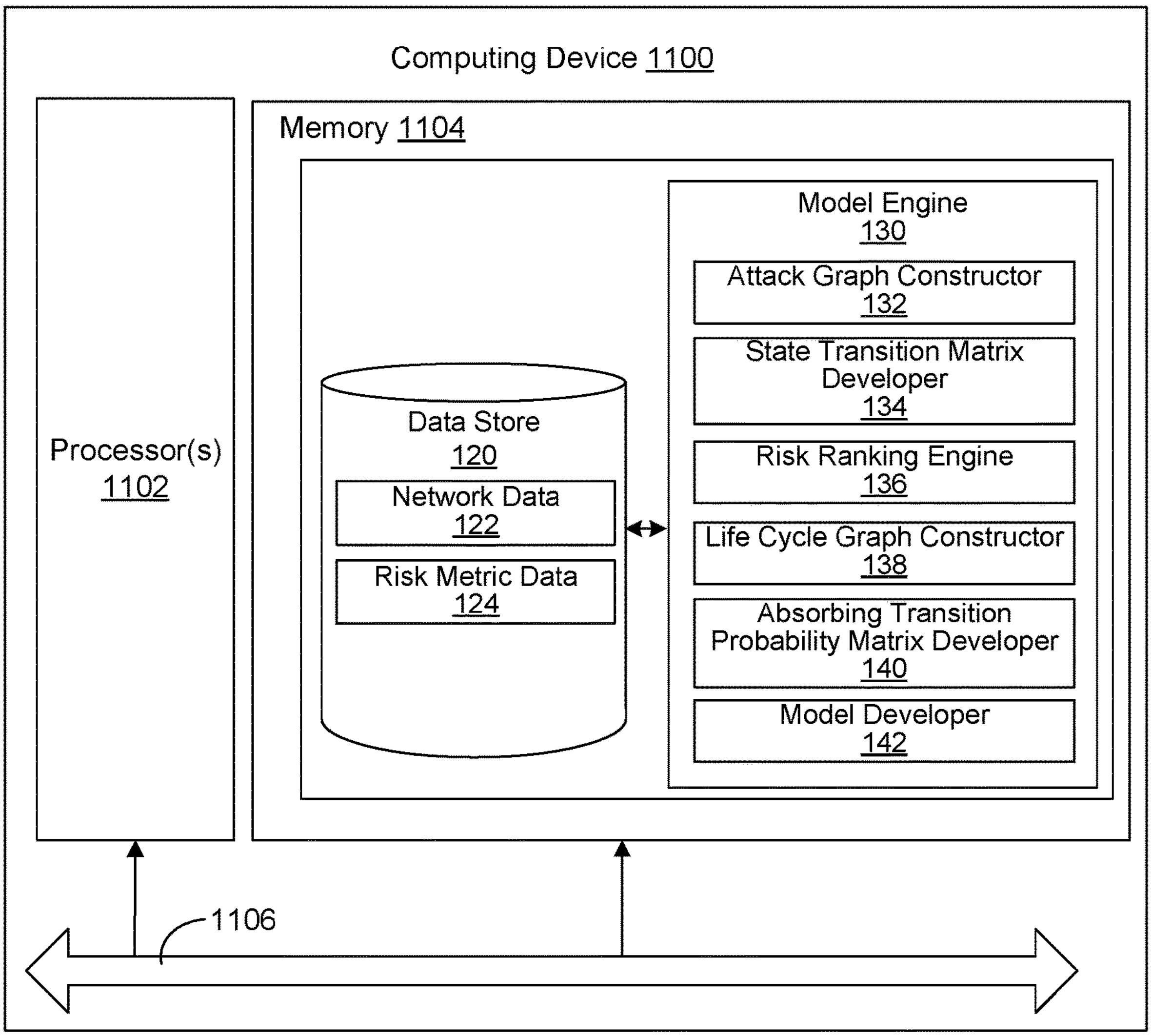

Computing Device 1100
Memory 1104
Model Engine 130
Attack Graph Constructor 132
State Transition Matrix Developer 134
Risk Ranking Engine 136
Life Cycle Graph Constructor 138
Absorbing Transition Probability Matrix Developer 140
Model Developer 142
Data Store 120
Network Data 122
Risk Metric Data 124
Processor(s) 1102
1106

FIG. 11

NON-LINEAR STOCHASTIC MODELS FOR PREDICTING EXPLOITABILITY

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 62/448,642, filed Jan. 20, 2017, the entire contents of which is hereby incorporated herein by reference.

BACKGROUND

In computing systems, a vulnerability can be defined as a weakness in software, hardware, firmware, etc. that can be exploited to gain access to certain resources. The management of vulnerabilities includes the practice of identifying and classifying vulnerabilities in computing systems and removing them. A vulnerability for which a working and implemented attack is known can be described as an exploitable vulnerability. A vulnerability is exploitable from the time when it is introduced to when it is removed or patched.

Vulnerabilities can be relatively difficult to categorize and mitigate. The Common Vulnerability Scoring System (CVSS) provides a way to characterize or define the principal characteristics of a vulnerability. The CVSS also provides a numerical score that reflects the severity of various vulnerabilities. The numerical score can be presented as a qualitative representation (e.g., low, medium, and high risk) to help prioritize vulnerability management processes.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the embodiments and the advantages thereof, reference is now made to the following description, in conjunction with the accompanying figures briefly described as follows:

FIG. **1** illustrates organizational aspects of the Common Vulnerability Scoring System (CVSS) framework according to various examples described herein.

FIG. **2** illustrates a computing environment for the generation of a predictive security model according to various examples described herein.

FIG. **3**A illustrates an example host access attack graph according to various examples described herein.

FIG. **3**B illustrates the example host access attack graph shown in FIG. **3**B along with an additional node related to an attacker in the system according to various examples described herein.

FIG. **4** illustrates two nodes from a host access attack graph according to various examples described herein.

FIG. **5** illustrates an example computing network scenario under evaluation according to various examples described herein.

FIG. **6** illustrates an example host access attack graph for the network shown in FIG. **5** according to various examples described herein.

FIG. **7**A illustrates a process for a predictive model for security risk evaluation according to various examples described herein.

FIG. **7**B illustrates a risk ranking algorithm in the process for the predictive model shown in FIG. **7**A according to various examples described herein.

FIG. **8** illustrates an example vulnerability life cycle graph including five states according to various examples described herein.

FIG. **9** illustrates an example risk factor behavior as a function of time according to various examples described herein.

FIG. **10** illustrates a process for developing a non-linear model for predicting exploitability according to various examples described herein.

FIG. **11** illustrates an example schematic block diagram of a computing device for the computing environment shown in FIG. **2** according to various embodiments described herein.

The drawings illustrate only example embodiments and are therefore not to be considered limiting of the scope of the embodiments described herein, as other embodiments are within the scope of the disclosure.

DETAILED DESCRIPTION

To protect network-accessible resources from attacks, various Intrusion Detection Systems (IDSs) are available. These intrusion detection and prevention based tools can provide signals to alert network administrators of intrusions, providing them with a picture of activities on the network. One important challenge for such IDSs is to develop mechanisms to aggregate the security risk of all systems in a network, evaluate the overall security risk for the systems, and present meaningful feedback and suggestions to network administrators.

To evaluate the security risk of a large scale enterprise network of computing systems, an administrator should consider not only single vulnerability exploits but also multi-stage and multi-host vulnerability exploits. To account for such multi-stage vulnerabilities, a host access attack graph can be relied upon to examine the logical relationships between multiple exploits. However, when the size and complexity of enterprise networks increase, two major problems occur. First, the host access attack graphs grow exponentially as the size of the networks increase in complexity. Second, the ability to evaluate the information conveyed in the host access attack graphs becomes more and more difficult. To help with those problems (and others in the field), recent studies have developed some useful statistical models that predict security risks based on various vulnerabilities using the Common Vulnerability Scoring System (CVSS) framework with a Markovian process.

The CVSS framework provides an open framework for communicating and analyzing the characteristics and impacts of vulnerabilities in computing systems. The quantitative model of the CVSS framework leads to repeatable and accurate measurements while enabling users to see the underlying vulnerability characteristics used to generate vulnerability-related scores. Thus, the CVSS framework is suitable as a standard measurement system for industries, organizations, and governments to accurately and consistently analyze vulnerabilities. Two common uses of the CVSS framework are the prioritization of vulnerability remediation activities and the calculation of the severity of vulnerabilities. The National Vulnerability Database (NVD) provides CVSS scores for almost all known vulnerabilities.

Risk metrics in the CVSS framework are composed of a number of metric groups, such as base, temporal, and environmental metrics, among others. Base metrics are constant over time across user environments and are related to the intrinsic characteristics of vulnerabilities. Base metrics include exploitability and impact metrics. The exploitability metrics are related to the ease and technical means by which a vulnerability can be exploited. The impact metrics are related to the consequences that can occur to components

after a successful exploit. For example, while a vulnerable component can be a software application, module, driver, etc., the impacted component can be a different software application, hardware device, or network resource. Temporal metrics are related to the characteristics of a vulnerability that change over time but not across environments. Environmental metrics are related to the characteristics of a vulnerability that are unique to a particular user environment (but might not change over time).

The values of base metrics can be assigned by an analyst, determined by a base metric score equation, determined by an equation and adjusted by an analyst, or calculated in other ways. A base metric can be computed as a score ranging from 0.0 to 10.0, for example, but other ranges can be used. An example equation to calculate a base metric score can be formed as two sub equations, for example, such as an exploitability sub-score equation for the exploitability sub score and an impact sub-score equation for the impact sub score. Base metric scores can be refined by the temporal and/or environmental metric scores in some cases to more accurately reflect risks posed by vulnerabilities in certain environments and/or over time.

A vulnerability is a flaw that exists in a computing system that can be exploited by one or more threats. In the context of vulnerabilities, a software vulnerability is an instance of an error in the specification, development, or configuration of software such that its execution can violate a security policy. Attackers normally use known vulnerabilities listed publicly on the NVD to penetrate computing systems. In some cases, attackers can leverage vulnerabilities that have not been disclosed publicly, called zero day vulnerabilities. Zero day vulnerabilities remain unknown to vendors, and such vulnerabilities gives attackers a “free pass” to attack certain hosts.

Attackers often penetrate computer networks via a chain of exploits, where each exploit in the chain creates the foundation for an upcoming exploit. A combination (e.g., chain) of such exploits is called an attack path, and a collection of attack paths can be used to develop an attack graph. Thus, a attack graph is representative of all known paths through which an attacker can infiltrate and attack a system. Various algorithms have been developed to construct attack graphs. However, it is relatively difficult to analyze networks using attack graphs, particularly as the number of nodes and complexity of networks increase. As the scalability and complexity of networks increase, the computational costs needed to create and evaluate attack graphs also increases. At the same time, without complicated attack graphs, it might not be possible to analyze the vulnerabilities in complex computing systems.

According to the embodiments described herein, a stochastic model is proposed for the evaluation of security risks in networks. Among other modelling data, the model uses exploitability and impact sub-scores of the CVSS framework. As described in further detail below, an example network having three host servers, each including one vulnerability, is considered. Based on the network architecture and vulnerabilities of the example network, a host access attack graph is constructed. From the host access attack graph, a state transition probability matrix is computed using exploitability and impact sub-scores. Using the Markovian random walk, the risk associated with each node is prioritized by ranking. Finally, the risk associated with all the nodes present in the network is summed, and the overall network security risk is determined. This quantitative value can be taken as a security metric to determine the risk of an entire network.

Further, new types of attack graphs can be relied upon among the embodiments. For example, a multiple layer attack graph can include upper and lower layers. The upper layer can include a host access attack graph and the lower layer can include host pair attack graphs. The lower level can describe the detailed attack scenarios between each host pair, and the upper level can show the direct network access relationship between each host pair. According to aspects of one embodiment, the stochastic models described herein can be based on upper layer attack or host access attack graphs.

In another embodiment, a new “risk factor” concept of vulnerability can be calculated as a function of time. In that concept, a Markovian approach can be introduced to estimate the probability of a particular vulnerability being at a particular “state” of a vulnerability life cycle. Here, those concepts and models are further developed using available data sources in a probabilistic foundation to enhance reliability. Other useful modeling strategies for vulnerability risk estimation are also introduced. For example, a new set of non-linear statistical models is presented. The models can be used in estimating the probability of being exploited as a function of time.

System administrators work to understand attackers and attack strategies to more effectively defend against attacks. To defend against attacks, it is helpful to develop a proper understanding of the risk associated with a given vulnerability. Having an effective model that predicts the risk of a given vulnerability being exploited as a function of time would be helpful to plan and implement security measures, allocate relevant resources, and defend systems accordingly. According to the embodiments described herein, the Markovian approach to vulnerability life cycle analysis is further developed to arrive at better modeling techniques to evaluate the “risk factor” using probability and statistical methods.

Thus, procedures to identify the probabilities for different states in a vulnerability life cycle are described. The probabilities are used to develop a number of statistical models to evaluate the risk factor of a particular vulnerability at time “t”. An absorbing transition probability matrix of all states of a particular vulnerability as a function of time is also described. A Markov chain process can be iterated to reach a steady state of the absorbing transition probability matrix, with the initial probabilities reaching the absorbing states, including exploited and patched states. A risk factor is also introduced for use as an index of the risk of a vulnerability being exploited. Finally, statistical models that can calculate the risk factor more conveniently without going through the Markovian process are described.

In one embodiment, a logical and approach is used to assign initial probabilities for each state of vulnerability. Additionally, the initial probabilities for each state of a vulnerability life cycle can be further refined based on certain logical assumptions. For example, CVSS scores can be used, but the initial probabilities can be refined or further calculated by taking the Common Vulnerabilities and Exposures (CVE) database (http://www.cvedetails.com/), at least in part, into consideration.

Additionally, using the methods described herein, three new statistical models are developed for vulnerabilities that differ based on their vulnerability score. Using these models, a user can estimate the risk of a particular vulnerability being exploited at time “t” and observe the expected behavior of the vulnerability throughout its life cycle.

FIG. **1** illustrates organizational aspects of the CVSS framework. CVSS is the open framework that provides quantitative scores representing the overall severity and risk of known vulnerabilities. A CVSS score can fall on a scale

from 0 to 10, for example, and consists of three major metrics, including base, temporal, and environmental as shown in FIG. **1**. Vulnerabilities with a base score range from about 0-3.9 can be considered relatively low vulnerability, 4.0-6.9 can be considered relatively medium vulnerability, and 7.0-10 can be considered relatively high vulnerability.

The base score can be computed using a number of sub-scores, such as the exploitability and impact sub-scores shown in FIG. **1**. The exploitability sub-score can be computed based on a combination of the access vector (AU), access complexity (AC), and authentication (AU) sub-scores. Further, the impact sub-score can be computed based on a combination of the confidentiality (C), integrity (I), and availability (A) sub-scores.

A Markov chain is one modeling technique that has been used effectively in various fields, such as reliability analysis, performance analysis, dependability analysis, and cybersecurity analysis, among others. As described below, the host access attack graph can be modeled using a Markov chain with the real behavior of the attacker in conjunction with the Markovian properties.

Mathematically, a Markov chain can be defined as a discrete stochastic process. More specifically, let S be a set of states (e.g., nodes in a host access attack graph). A Markov chain is a sequence of random variables X_0, X_1, X_2, . . . $X_n \in S$ that satisfies the "Markovian property":

$$P[X_{n+1}=y|X_0=x_0, X_1=x_1, \quad . \quad . \quad . \quad X_n=x_n]=P[X_{n+1}=y|X_n=x_n]$$

The Markovian property reveals the fact that the transitions between states are memoryless and that transitioning to the next step depends only on the current state and not on any previous states. This property can be correlated with the behavior of an attacker in the sense that an attacker needs to exploit several nodes before reaching a goal node. When the attacker starts attacking an initial node to reach the goal node, there can be many other nodes, called intermediate nodes, before reaching the goal node. When an attacker reaches any intermediate node, there is no memory of previous nodes. The attacker launches further attacks until the goal node is found.

To advance the attack, an attacker can move from one intermediate node to another intermediate node. In the examples described herein, the selection of the best intermediate node depends on three parameters, including the exploitability sub-score, the impact sub-score, and the bias factor or skill of the attacker.

Without loss of generality, transition states are independent of time. Mathematically, there exists some state transition probability matrix, P(x,y), such that:

$$P(x,y)=P[X_{n+1}=y|X_n=x_n], \text{ for all } n.$$

A new set of states S×[n] can be created having a different set of states associated with each timestep. To simulate a Markov chain, a stochastic state transition probability matrix P(x,y) and an initial probability distribution is needed. The initial risk associated with each node in the host access attack graph can be considered an initial probability distribution as described in further detail below. Once the stochastic state transition probability matrix P(x,y) and initial risk are determined, then the risk of the entire network can be determined utilizing the basic properties of the Markovian process.

FIG. **2** illustrates a computing environment **110** for the generation of a predictive security model according to various examples described herein. Among other components, the computing environment **110** includes a data store **120** and a model engine **130**. Among other data, the data store **120** includes memory areas to store network data **122** and risk metric data **124**. The model engine **130** includes an attack graph constructor **132**, a state transition matrix developer **134**, and a risk ranking engine **136**, the operation of each of which is described in further detail below with reference to FIGS. **3**A, **3**B, **4**, **5**, **7**A, and **7**B. The model engine **130** further includes a life cycle graph constructor **138**, an absorbing transition probability matrix developer **140**, and a model developer **142**, the operation of each of which is described in further detail below with reference to FIGS. **8** and **10**.

The computing environment **110** can be embodied as one or more computing devices or systems. In various embodiments, the computing environment **110** can be embodied as a desktop, laptop, server or other type(s) of computing devices or systems. As described herein, the model engine **130** in the computing environment **110** is configured to generate a predictive security model. The model can be generated to evaluate relatively large networks of computing systems having a number of network nodes. The computing systems and devices in such networks can be located at a single installation site or distributed among different geographical locations. The computing devices in such networks can also include computing devices that together embody a hosted computing resource, a grid computing resource, and/or other distributed computing arrangement.

The computing environment **110** and the network of computing systems evaluated by the computing environment **110** can be coupled to one or more networks embodied by the Internet, intranets, extranets, wide area networks (WANs), local area networks (LANs), wired networks, wireless (e.g., cellular, 802.11-based (WiFi), bluetooth, etc.) networks, cable networks, satellite networks, other suitable networks, or any combinations thereof. The computing environment **110** can communicate with other computing devices and systems using any suitable systems interconnect models and/or protocols. Although not illustrated in FIG. **2**, the computing environment **110** can be coupled to any number of network hosts, such as website servers, file servers, network switches, networked computing resources, databases, data stores, and other network or computing platforms.

The network data **122** can include data related to the network of computing systems being evaluated by the model engine **130**. In that context, the network data **122** can define the types of network and computing devices and systems being evaluated by the model engine **130**, such as the serial numbers, model numbers, operating system versions, services, and other identifying information. The network data **122** can also specify the logical arrangement of those devices among each other, including the network connections between them. The network data **122** can include all the information necessary for the attack graph constructor **132** to generate a host access attack graph as described herein.

The risk metric data **124** can include a number of risk metrics associated with devices specified in the network data **122** including CVSS and CVE data. As one example, according to the CVSS framework, the risk metrics can include base, temporal, and environmental metrics, among others, for the devices specified in the network data **122**. However, the risk metric data **124** is not limited to the types of metrics used in the CVSS framework, as other types and formats of risk metrics can be relied upon.

The attack graph constructor **132** is configured to construct host access attack graphs based on the network data **122**. The network topology information defined in the network data **122** can include serial numbers, model numbers, operating system versions, services, and other identifying information. The network topology information can also specify the logical arrangement of host devices among each other, including the network connections between them. The network topology information can specify a number of hosts in enterprise systems, services running on each host in the network, rules defined on firewalls, network switches, etc., and vulnerabilities associated with each host and service among other topology information. For simplicity, a limited number of nodes are present in the examples described herein, but attack graphs of any size can be used. In the attack graphs described herein, each node can be representative of any of the above-described (or related) types of host computing devices, systems, or services. Each host can include various types of vulnerabilities. Example attack graphs are shown in FIGS. **3**A and **3**B and described below.

Once an attack graph is created, scores can be assigned to the vulnerabilities of the hosts in the attack graph using information from the risk metric data **124**, such as CVSS framework metric data. The scores can be computed based on a number of scores and sub-scores, such as those shown in FIG. **1**, for example, using with one or more expressions, equations, or sub-equations that relate them. In some cases, one or more standard expressions can be used calculate scores based on matrices that provide a quantitative score to approximate the ease and/or impact of the vulnerabilities in the nodes. The exploitability and impact sub-scores, for example, can also be combined to provide the basis of assigning scores to directed connections among the nodes in attack graphs as probabilities. Those probabilities can represent the possibility of a vulnerability being exploited by an attacker.

To implement the stochastic model, the behavior of the attacker should also be considered. As one example, it can be assumed that the attacker would choose a vulnerability that maximizes the chances of success in the goal. In one example, if the attacker terminates attacking for any reason, then the model can move the attacker back to the initial state. Finally, utilizing the properties of a Markov chain, the risk of one or more individual nodes can be computed. The nodes are then prioritized based on risk, and the risks of all the nodes are summed to give the total security risk present in the computing system environment.

FIG. **3**A illustrates an example host access attack graph for a networked computing environment according to various examples described herein. In FIG. **3**A, S_i, i=1, 2, 3, . . . g are host nodes and S_g is a goal node. The host access attack graph shown in FIG. **3**A is a representative example, and host access attack graph can be constructed to any size necessary and with any number of intermediate layers.

A node in the host access attack graph is representative of a computing device or system in the networked computing environment. Each node can be representative of a different type of computing device, such as a server, desktop, laptop, handheld, or other type of computing system. The nodes can also be representative of other types of network devices including network switches, routers, firewalls, and other devices. In some cases, the nodes can also be representative of one or more software services executing on one or more computing devices.

A directed connection (e.g., arrowed line) between two nodes represents the access relationship between the nodes. In FIG. **3**A, a directed connection from host S_1 to host S_2 represents the access available on S_2 from S_1, and the same is applicable to other hosts. In the example shown, there is one directed connection from any given node to any other node. As such, the host access attack graph shown in FIG. **3**A does not include multiple connections between the same two nodes. In other cases, the host access attack graph can include a number of connections between two or more nodes. In one example, the model retains only the highest access achieved between hosts, because higher levels of access to the destination or goal host mean more powerful attacks can be achieved.

Once the host access attack graph is constructed by the attack graph constructor **132**, then the basic foundation is developed for further analysis by the state transition matrix developer **134** and the risk ranking engine **136**. To make the host access attack graph more applicable and realistic, an additional dummy node is added as shown in FIG. **3**B to represent the attacker. The attacker can start by exploiting an immediate node by gaining a high level of privileges, for example. In the proposed model, an attacker starts attacking an immediate node and continues to launch attacks until the attacker reaches a goal node.

However, even if an attacker is equipped with sophisticated tools and a high level of experience, there is no guarantee that the attacker will reach the goal node. This can happen if the attacker is unable to exploit a certain vulnerability, the attacker is discovered by an intrusion response team, or other circumstances. According to one case for the model, when the attacker stops launching attacks at any point for any reason, the attacker goes back to the initial state from where the attack began. To incorporate this attack scenario, the attacker node A is introduced as shown in FIG. **3**B. Thus, in FIG. **3**B, node A represents an attacker, and there is a directed connection from every node to node A. This implies that when the attacker gives up exploiting the node for any reason, the attacker goes back to the initial state and proceeds to search for alternative options. From any node S_i, this return directed connection can be defined as (S_i, A).

The state transition matrix developer **134** is configured to develop a state transition probability matrix based on exploitability scores and impact scores, for example, associated with individual nodes in the host access attack graph. The state transition probability matrix defines certain probability metrics related to the order in which nodes in an attack graph are likely to be attacked. In one case, it can be assumed that the decisions of an attacker depends on two parameters. The first parameter is exploitability, related to the level of complexity involved to attack a given node. The second parameter is impact, related to how much impact an attacker can make when the node is exploited. The CVSS framework provides numerical scores for those parameters, where 0 signifies the most secure and 10 signifies the least secure. These two parameters can be conceptually expressed by:

$$\text{ExploitabilityBenefit}=f(\text{Exploitability},\text{Impact}). \tag{1}$$

In Equation 1, ExploitabilityBenefit is defined as a function of the exploitability and impact metrics. Based on these values, the state transition matrix developer **134** can estimate how an attacker would determine or consider the benefit to move from one node to another.

To clarify this concept, consider any two nodes from the host access attack graph shown in FIG. **4**, where S_j and S_k are nodes j and node k, respectively, with V_i and V_k being the corresponding vulnerabilities. In FIG. **4**, there is a directed connection from node j to node k. The value of Exploit-

abilityBenefit in the case of FIG. **4** is representative of whether an attacker would decide to move from node j to node k.

The decision to move from one node to another node can depend not only on the exploitability and impact factors, but also on the skills and expertise of the attacker. The state transition matrix developer **134** can account for the skills and expertise of an attacker using a bias factor, which can vary in value from attacker to attacker. Incorporating all three parameters (i.e., exploitability, impact, and bias), Equation 1 can be extended to:

$$a_{jk}=\beta\cdot \mathrm{Exp}(v_k)+(1-\beta)\cdot \mathrm{impact}(v_k),\ \text{where } 0<\beta\leq 1. \tag{2}$$

In Equation 2, a_{jk} is the ExploitabilityBenefit score to move from node j to node k, $\mathrm{Exp}(v_k)$ is a function that measures the level of difficulty in exploiting node k, and Impact (v_k) is a function that measures the potential damages or losses that occur due to a successful exploitation of node k. The quantitative value related to the level of difficulty can be provided by CVSS, and the quantitative value related to the potential damages or losses can be also provided by CVSS.

The possibility that exploitation might occur depends on the experience and skills of the attacker. To account for that factor, the bias factor β can range from 0 to 1 based on the level of experience and skill of the attacker. When the exploitability and impact scores are combined with their corresponding bias factors, a weighted value (i.e., the ExploitabilityBenefit, a_{jk}) is obtained to quantify the benefit for an attacker to move from node j to node k.

To move the attacker from an initial node to a goal node, the attacker may need to penetrate several intermediate nodes. Assuming j is an initial node, g is a goal node, and three intermediate nodes k, **1**, and m, one possibility is that the attacker reaches the goal node by exploiting node j to node k, node k to node **1**, node **1** to node m, and finally node m to node g. Thus, the state transition matrix developer **134** is configured to develop a weighted adjacency matrix A, such as:

$$A=\begin{bmatrix} a_{00} & a_{01} & \dots & a_{0g} & \dots & a_{0n} \\ a_{01} & 0 & \dots & a_{1g} & \dots & a_{1n} \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ a_{n0} & a_{n1} & \dots & a_{ng} & \dots & 0 \end{bmatrix}$$

Each element of the adjacency matrix A can be computed by the state transition matrix developer **134** using Equation 2. Diagonal values of the adjacency matrix A are all zero because no cost is involved to move from the current node to itself. The elements of the adjacency matrix A are not normalized, and the state transition matrix developer **134** can convert the non-normalized values into probabilities using Equation 3. Equation 3 defines that, in each step, the attacker goes from node j to k with a probability given by:

$$p_{jk}=\frac{A(j,k)}{\sum_l A(j,l)}. \tag{3}$$

Writing Equation 3 in matrix form:

$$P=DA, \tag{4}$$

where, A is the weighted adjacency matrix, P is the state transition probability matrix that provides the transition probability that the attacker moves from one state to another state, and D is the diagonal matrix computed using Equation 5 below.

$$D_{jk}=\begin{cases} \dfrac{1}{\sum_l A(j,l)} & \text{if } j=k \\ 0 & \text{Otherwise} \end{cases}. \tag{5}$$

Thus, using Equations (1)-(5), the state transition matrix developer **134** can develop a state transition probability matrix representative of the probability that an attacker moves from one state to another state in the host access attack graph constructed by the attack graph constructor **132**.

The risk ranking engine **136** is configured to rank the risk associated with the nodes in the host access attack graph constructed by the attack graph constructor **132** with reference to the state transition probability matrix developed by the state transition matrix developer **134**. The risk analysis is based on the relative rank value for every node of the host access attack graph. In this context, R is the risk vector and its initial risk value is computed based on the number of hosts present in the host access attack graph. If N nodes exist in the host access attack graph, then all the node ranks can equal 1/N. This initial risk is first injected by the starting node of an attacker. Risk values flow, level by level, until convergence. The risk ranking and total risk calculation process is described in further detail below with reference to FIG. **7**B.

The risk value of r_k for a node k depends upon the rank of its parent nodes. The risk value of the node set by the initial node represents the starting node of the attacker. When the ranking process is started, the intermediate risk value or values are computed via iteration. The intermediate values will flow, level by level, until a steady state is achieved. Mathematically, if r_k is the risk of node k given in the host access attack graph, then the risk ranking engine **136** can compute the risk of node k using Equation 6, by:

$$r_k=\sum r_k p_{jk}. \tag{6}$$

Suppose, $R=(r_1, r_2, r_3, \dots r_z)$ is the risk vector, where r_j is the rank of node j. In that case, Equation 6 can be further extended to Equation 7 as shown below. The risk values are normalized, where $0\leq r_k\leq 1$ for all j, and

$$\sum r_k=1.$$

Thus, written in matrix form, the risk vector R is given by R times the state transition probability matrix P, by:

$$R=RP. \tag{7}$$

The value of R in Equation 7 is recursive and must be iteratively calculated until convergence, which is expressed by Equation 8 as:

$$R^t=R^{t-1}P. \tag{8}$$

The risk ranking engine **136** is configured to evaluate the risk in the attacking process based on the Markovian random

walk, a condition for the iterative computation to converge. The probability distribution of risk analysis of the host access attack graph after the attacker follows one link in the graph is $R^1=RP$, where R is the risk vector and P is the one step state transition probability matrix identified by Equation 4. Similarly, after two links, the probability distribution is $R^2=R^1P$. Assuming this iteration converges to a steady state probability, then we have $R^t=R^{t-1}P$, where R^t is an eigenvector of P.

To validate the proposed stochastic model, a network environment **200** is shown in FIG. **5**. The network environment **200** includes a number of target hosts, including a publicly accessible web server **202**, a publicly accessible file server **204**, and a backend database server **206** of a network **220**. An attacker **210** is located outside the network **220**. Packet transmissions are controlled via two firewalls, including an external firewall **230** and an internal firewall **232**. The external firewall **230** allows any packet to be transmitted to the web server **202** and the file server **204** from outside the network **220**, but the backend database server **206** cannot be directly accessed from outside the network **220**. The internal firewall **232** manages the transmission of packets within the network **220**.

Rules are created for the firewalls **230** and **232** to filter inbound and outbound traffic. A summary of the rules of the firewalls **230** and **232** are shown in Table 1 below.

TABLE 1

Source	Destination	Service	Action
All	Web Server	http	Allow
All	Web Server	ftp	Allow
All	File Server	ftp	Allow
Web Server	Database	oracle	Allow
File Server	Database	ftp	Allow
All	All	All	Deny

Each of the target hosts in the network shown in FIG. **5** includes a single vulnerability. An attacker can utilize the vulnerability to compromise the host. The vulnerabilities are shown in Table 2 below, along with the exploitability and impact sub-scores for each from the NVD.

TABLE 2

Host	Vulnerability	CVE-ID	Score	Impact Sub-Score	Exploitability Sub-Score
Web Server	Apache Chunked Code	CVE-2002-0392	7.5	6.4	10
File Server	Wuftpd Sockprintf	CVE-2003-1327	9.3	10	8.6
Database	Oracle Tns listener	CVE-2012-1675	7.5	6.5	10

FIG. **6** illustrates a host access attack graph for the network shown in FIG. **5**. The attacker **210**, web server **202**, file server **204**, and backend database server **206** are designated M_0, M_1, M_2, and M_3, respectively. The connections from all the nodes to the attacker node M_0 are omitted to view the graph more clearly.

Applying Equation 2 on the host access attack graph shown in FIG. **6**, the weighted adjacency matrix A is:

$$A = \begin{bmatrix} 0 & 8.2 & 9.3 & 0 \\ 1 & 0 & 9.3 & 8.2 \\ 1 & 8.2 & 0 & 8.2 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

The value of the biased factor β is assumed to be 0.5 for the example. When the attacker **210** stops attacking due to any unusual circumstances, then the attacker **210** will return to the initial node M_0. Hence, the elements of the first column of the weighted adjacency matrix are 1. In other words, the weights of the connections from all host nodes to the attacker node M_0 are considered **1**, a sure event. The other elements of the weighted adjacency matrix A are calculated using Equation 2. For example, the entry of the first row and second column is (0.5×10+0.5×6.4)=8.2. This is the weighted value of the benefit for the attacker **210** to move from node M_0 to node M_1. The other elements of the weighted adjacency matrix A can be determined similarly.

After the weighted adjacency matrix A is calculated, the elements can be converted into respective probabilities. The entries of the main diagonal are obtained using Eq. 5, as:

$$D = \begin{bmatrix} 0.05714 & 0 & 0 & 0 \\ 0 & 0.05405 & 0 & 0 \\ 0 & 0 & 0.05747 & 0 \\ 10 & 0 & 0 & 1 \end{bmatrix}$$

An element of the first row and the first column of the diagonal matrix is determined as 1/(8.2+9.3)=0.05714, and the other elements can be calculated similarly. Using the weighted adjacency matrix A and the diagonal matrix D as shown above, a state transition probability matrix P can be obtained using Equation 4, as:

$$P = \begin{bmatrix} 0 & 0.46857 & 0.5314 & 0 \\ 0.0540 & 0 & 0.5027 & 0.4432 \\ 0.0575 & 0.4712 & 0 & 0.4713 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

The value of the first row, second column is 0.46857. That value is representative of the probability that the attacker would move from node M_0 to node M_1. The other values of in the state transition probability matrix P are representative of the probability that the attacker would move between other nodes (or desirability for the attacker to do so).

The host access attack graph shown in FIG. **6**, includes four nodes. Based on the risk ranking algorithm, if there are four nodes then 1/4=0.25 is the initial risk of each node, hence an initial risk vector of R=(0.25, 0.25, 0.25, 0.25). When the initial risk vector R and the state transition probability matrix P are iteratively multiplied using Equation 8, convergence is achieved to the values listed in Table 3 below.

TABLE 3

Node	Risk
M_0	0.2609688
M_1	0.2455964
M_2	0.2620094
M_3	0.2314254

From Table 3, it is clear that node M_2 is more risky than nodes M_1 and M_3. Thus, the vulnerability of the file server should be patched before those of the other nodes. Further, the total sum of the risk associated with nodes M_1, M_2, and M_3 is as 0.74. This value can be used as a security metric revealing the fact that the network shown in FIG. **6** is not very secure and appropriate actions should be taken.

FIG. **7**A illustrates a process for a predictive model for security risk evaluation, and FIG. **7**B further illustrates a risk ranking algorithm in the process shown in FIG. **7**A. The process flowcharts in FIGS. **7**A and **7**B can be viewed as depicting example steps performed by the computing environment **110**, although other computing systems and environments can perform the process. The flowcharts in FIGS. **7**A and **7**B provide merely one example of a functional sequence or arrangement of steps that can be employed to implement the processes for predictive modeling and risk ranking described herein. Although the processes are described in connection with the computing environment **110**, other computing environments, systems, and/or devices can perform the processes. Additionally, although not explicitly stated below, among each of the process steps described, any number of intermediate data accessing, storing, and logging steps can be performed.

Turning to FIG. **7**A, at step **302**, the process can include the attack graph constructor **132** constructing a host access attack graph. The host access attack graph can be constructed based on data stored in the network data **122**, for example, according to characteristics of a network of computing systems. The host access attack graph can include a plurality of nodes such as those shown in FIG. **3**A or **3**B.

At step **304**, the process can include the state transition matrix developer **134** gathering security or vulnerability metrics related to one or more of the nodes in the host access attack graph. The metrics may be gathered from the risk metric data **124** or from another computing system via network communications. As one example, the state transition matrix developer **134** can gather exploitability scores and impact scores associated with the nodes in the host access attack graph. The exploitability and impact scores can be CVSS scores or other scores developed according to another vulnerability scoring system.

At step **306**, the process can include the state transition matrix developer **134** developing a state transition probability matrix based on the scores gathered at step **304** and the host access attack graph constructed at step **302**. In one example, the state transition matrix developer **134** can develop the state transition probability matrix according to Equations (1)-(5) as described above with reference to the exploitability scores and the impact scores.

At step **308**, the process can include the risk ranking engine **136** evaluating and ranking risk associated with the nodes in the host access attack graph constructed at step **302** with reference to the state transition probability matrix developed at step **306**. The process of evaluating and ranking the risk is illustrated in further detail in FIG. **7**B.

Turning to FIG. **7**B, the risk ranking process includes creating a risk vector with initial risk values at step **310**. As described above, a risk vector R and its initial risk values can be computed based on the number of hosts present in the host access attack graph. If N nodes exist in the host access attack graph, then the rank of all nodes can be equal to 1/N.

At step **312**, the process includes the risk ranking engine **136** iterating the risk vector from step **310** with the state transition probability matrix developed at step **306**. When the ranking process is started, the intermediate risk value or values are computed via iteration. The intermediate values will flow, level by level, until a steady state is achieved according to Equations (6)-(8) above.

At step **314**, it is assumed that the iterating at step **312** has converged, and the risk vector includes a number of risk elements, each representative of the risk of a respective node in the host access attack graph. Using this converged risk vector, the process can include the risk ranking engine **136** prioritizing the risk associated with each node at step **316** by ranking them based on the level of risk of each. In other words, a node associated with a higher level of risk can be prioritized for remediation over a node associated with a relatively lower level of risk.

Finally, at step **318**, the process can include the risk ranking engine **136** computing a total risk for the network of computing systems being evaluated. The total risk can be calculated based on a total risk for all the elements in the risk vector, for example. Thus, the risks of all the nodes are summed to give a total security risk present in the network of computing systems.

Thus, as described above, a stochastic model is developed for cybersecurity using a host access attack graph to determine the overall network security risk. The model uses Markov chains in conjunction with CVSS framework metrics to analyze risks associated with structures of various networks. The model can be used to identify critical nodes in the host access attack graph where attackers may be most likely to focus. Based on that information, a network administrator can make appropriate, prioritized decisions for system patching. Further, a flexible risk ranking technique is described, where the decisions made by an attacker can be adjusted using a bias factor. The model can be generalized for use with complicated network environments.

Turning to other embodiments, FIG. **8** illustrates an example vulnerability life cycle graph including five states according to various examples described herein. A vulnerability life cycle graph for a given vulnerability has several state nodes, each representative of a state of a vulnerability in a computing system. The life cycle graph shown in FIG. **8** can be constructed by the life cycle graph constructor **138** of the computing environment **110** shown in FIG. **2**, for example. The life cycle graph can be generated based on the network data **122** and/or the risk metric data **124**.

The logical probabilities for a hacker to reach each state in the life cycle graph shown in FIG. **8** can be assigned to each state by the life cycle graph constructor **138** by examining the properties of specific vulnerabilities in the computing system or systems being modeled. The vulnerability life cycle graph shown in FIG. **8** has two absorbing states, called the patched state and the exploited state. These states allow the vulnerability life cycle graph to be modeled as an absorbing Markov chain. FIG. **8** shows states three and five as the absorbing states of the life cycle graph, where state three is the "exploited" state and state five is the "patched" state.

λ_i is defined as the probability of transferring from, state i to state j, where, i, j=1, 2, 3, 4, 5. In actual situations, the probability of discovering a vulnerability can be assumed to be very small. Therefore, as a starting point, a small value

can be assigned for λ_1. The probabilities for λ_1, λ_2, λ_3, λ_4, λ_5 can then be assigned accordingly. Checking for several random values of λ_i, the behavior of the other states can be observed as a function of time. Using the transition probabilities, the absorbing transition probability matrix can be derived, which follows the properties defined under the Markov chain transformation probability method.

However, instead of randomly assigning transition probabilities for each state presented in the life cycle, a new set of probabilistically reliable methods is described. Because it can be challenging to acquire a complete set of information relevant to the vulnerabilities of a system, available and reliable data resources about vulnerabilities can be used to develop a new methodology described below.

As discussed above, CVSS is a standard for assessing the magnitude of information system vulnerabilities. The CVSS framework provides an open framework for communicating and analyzing the characteristics and impacts of vulnerabilities in computing systems. The quantitative model of the CVSS framework leads to repeatable and accurate measurements while enabling users to see the underlying vulnerability characteristics used to generate vulnerability-related scores. Thus, the CVSS framework is suitable as a standard measurement system for industries, organizations, and governments to accurately and consistently analyze vulnerabilities. As examples, the CVSS framework can be used to calculate the severity of vulnerabilities and prioritize vulnerability remediation activities. In extension to the CVSS framework, the NVD database provides CVSS scores for almost all known vulnerabilities.

The NVD database provides base metric stores for vulnerabilities only because the temporal and environmental scores vary based on other factors related to the organization that uses the computer system. The base score for more than 75,000 different vulnerabilities can be calculated using 6 different matrices. It is managed by the Forum of Incident Response and Security Teams (FIRST).

As introduced above, CVSS establishes a standard measure of how much concern a vulnerability warrants, compared to other vulnerabilities, so that efforts can be prioritized. The base metric scores can range from 0 to 10, where base scores in the range 7.0-10.0 are high, 4.0-6.9 are medium, and 0-3.9 are low. Thus, the absorbing transition probability matrices and statistical models described herein are based on this classification of the CVSS score. Instead of randomly assigning a probability (i.e., each λ_i) for each state, the CVE resources can be used for probabilistically estimating reliable values for each state. The approach in assigning initial probabilities into each state of the life cycle is described in further detail below.

Once the vulnerability life cycle graph is constructed, the absorbing transition probability matrix developer **140** is configured to assign initial probabilities (i.e., each λ_i) for different states in the life cycle graph. Table 1 presents the initial probabilities used in the example described herein. Developing or estimating these initial probabilities can require a significant amount of data resources. To estimate λ_1, for example, one may include any number of the total number of vulnerabilities in each category of a computing system, each ranging from 0 to 10 in magnitudes, along with information related to their discovery with respect to time. Similarly, for the other states, the number of vulnerabilities discovered, exploited before disclosed, exploited after discovery but before patched, patched before disclosure, and patched after disclosure can be relied upon under each CVSS score level.

To start, the absorbing transition probability matrix developer **140** is configured to reference the CVSS scores available for each category of vulnerability. However, CVSS scores may not be available to provide data for all cases. Thus, the absorbing transition probability matrix developer **140** is further configured to reference the CVSS classifications available in the CVE detail website where available, and other resources of information can be used in some cases. Examples of the other resources include industry reports, thesis papers, and other resources of data on vulnerabilities.

TABLE 4

States Represented by the Transition Probabilities in the Vulnerability Life Cycle

Probability - λ_i	State Represented
λ_1	Discovered
λ_2	Exploited before patched or disclosed
λ_3	Disclosed but not yet patched or exploited
λ_4	Patched before disclosed
λ_5	Exploited after disclosed
λ_6	Patched after disclosed

In one case, 75705 vulnerability scores were calculated according to their CVSS scores under each of the three categories to find the total number of vulnerabilities and exploits. Those scores are used to assign probabilities of discovery (λ_1) and exploited (λ_2) for each CVSS score level.

In the example provided herein, data from other resources was used to assign probabilities for the disclosed but not yet patched or exploited state (λ_3), the patched before disclosed state (λ_4), the exploited after disclosed (λ_5) state, and the patched after disclosed (λ_6) state.

To calculate an estimate for λ_1 (e.g., "the probability of a vulnerability is being discovered") for three categories of CVSS scores, it is helpful to have an estimate for the population of a "total number of (known and unknown) vulnerabilities at a particular time" to get the proportion of discovered vulnerability out of the total. But, at a given time, it may not be possible to know the total number of vulnerabilities in computing systems as the number of vendors, application software, system software, and other apps are undefined. Therefore, to have a logical estimate for the total number of vulnerabilities for each year, the absorbing transition probability matrix developer **140** is configured to estimate a cumulative number of vulnerabilities in the computing system being modeled. Then, the number of vulnerabilities discovered in a particular year as a proportion of the cumulative number of vulnerabilities in the next calendar year can also be calculated. Once these proportions considering the years from 1999 to 2015, for example, are calculated, the average of those proportions can be used as the estimate for λ_1.

When calculating λ_1, it can be assumed that a number of unknown vulnerabilities in a particular year are discovered in the next year and the accumulated number of vulnerabilities in a particular year is an estimate for the population size of the vulnerabilities in the previous year.

In one case, to calculate an estimate for λ_2 (e.g., "the probability of a particular vulnerability being exploited before patched or disclosed"), the data provided in the CVE website was used, although other sources can be relied upon. The entire set of exploited vulnerabilities were calculated for 10 different categories (or CVSS score levels) of interest. To

calculate an estimate for λ_3 (e.g., "the probability of a vulnerability being disclosed but not yet patched or exploited"), the equation $\lambda_3=1-(\lambda_2+\lambda_4)$ can be used.

To calculate an estimate for λ_4 (e.g., "the probability of a vulnerability being patched before disclosed"), the information available from other resources was used. To calculate an estimate for λ_5 (e.g., "probability of a vulnerability being exploited after disclosed" and for λ_6 (e.g., "probability of a vulnerability being patched after disclosed"), the information available from other resources was used. One example resource estimates that the probability of a vulnerability being exploited after being disclosed is greater than the probability of it being patched. The resource estimates that there is a probability of about 0.6 for a disclosed vulnerability being exploited. Thus, in one case, fixed values of 0.6 and 0.4, respectively, are used for λ_5 and λ_6.

Table 5 presents example probabilities for each state with respect to each category/level of vulnerability.

TABLE 2

Estimates of Transition Probabilities for each Category of Vulnerability

Vulnerability level	λ1	λ2	λ3	λ4	λ5	λ6
Low	0.1777	0.016303	0.183696615	0.8	0.6	0.4
Medium	0.1888	0.08104	0.118960089	0.8	0.6	0.4
High	0.1804	0.147552	0.052448328	0.8	0.6	0.4

Using these transition probabilities for each level, the absorbing transition probability matrix developer **140** is configured develop the absorbing transition probability matrix for a life cycle according to the Markov process described above.

Once a vulnerability life cycle graph is developed with two absorbing states and initial probability estimates for each state, the general form of the absorbing transition probability matrix for the vulnerability life cycle can be given as follows.

$$P_i(t)=\begin{bmatrix} 1-\lambda_1 & \lambda_1 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & \lambda_3 & \lambda_4 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \lambda_5 & 0 & \lambda_6 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

where $P_i(t)$ is the probability that the system is in state i at time t. For t=0 we have $P_1(0)=1$, the probability that the system is in state 1 at the beginning (t=0), and $P_2(0)=0$, $P_3(0)=0$, $P_4(0)=0$, $P_5(0)=0$.

Therefore, the initial probability can be given as [1 0 0 0 0], which covers the initial probabilities of each state of the vulnerability life cycle. Here, the initial state having a probability of one is representative of the fact that, at the initial time (t=0), the vulnerability has not yet been discovered. Therefore, the probabilities for all others states are zero.

Now, for three different categories of vulnerabilities, the absorbing transition probability matrix can be iterated by the absorbing transition probability matrix developer **140** using the Markovian process until the matrix reaches its "steady state". For t=0, $\overline{P^{(0)}}=[1\ 0\ 0\ 0\ 0]$. For t=1, results in $\overline{P^{(1)}}=\overline{P^{(0)}}P$. For t=2, $\overline{P^{(2)}}=\overline{P^{(0)}}P$. Thus, for =n, $\overline{P^{(n)}}=\overline{P^{(0)}}P^{(n)}$. Using this method, the probability is changing with time and is related to each "state," and it is possible to find the statistical model that can fit the vulnerability life cycle.

As an example, for the vulnerabilities in category one, where $\lambda_1=0.1777$, $\lambda_2=0.0163$, $\lambda_3=0.1837$, $\lambda_4=0.8$, as $\lambda_5=0.6$, and $\lambda_6=0.4$, the absorbing transition probability matrix is written as follows:

$$P=\begin{bmatrix} 0.8223 & 0.1777 & 0 & 0 & 0 \\ 0 & 0 & 0.0163 & 0.1837 & 0.8 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

As this algorithm is executed for the vulnerabilities of category one, the stationarity (steady state) was reached (to 4 decimal digits) at t=86. That is, the minimum number of steps so that the vulnerability reaches its absorbing states is 86 and the resulting vector of probabilities for each of the absorbing states is obtained as the output of the calculation process.

As shown below, the transition probabilities are completely absorbed into the two absorbing states which gives the "probability of the vulnerability being exploited" and the "probability of the vulnerability will be patched". All other states have reached the probability of zero.

$$P=\begin{bmatrix} 0.8223 & 0.1777 & 0 & 0 & 0 \\ 0 & 0 & 0.0163 & 0.1837 & 0.8 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}\rightarrow$$

$$P^{(86)}=\begin{bmatrix} 0 & 0 & 01524 & 0 & 0.8476 \\ 0 & 0 & 0.1524 & 0 & 0.8476 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\overline{P^{(86)}}=\overline{P^{(0)}}P^{(86)}=[0\ \ 0\ \ 0.1265\ \ 0\ \ 0.8735].$$

Thus, it will take the hacker 86 steps and a 12.7% chance to exploit the security system and 87.3% probability to reach the patched state. The estimate is that, after t=86, one of the two states will be reached.

Initially, the 3rd state was identified as "the state of being exploited" and the 5th state as "the state of being patched" in the vulnerability life cycle graph. Based on the current data resources available relevant to the vulnerabilities of category one, these results can be used as estimates for the probabilities of being exploited and being patched. The results from this Markovian model for the vulnerability life cycle shows that the sum of the resulting probabilities equals to one (0.1265+0.8735=1). In other words, this indicates that the model estimates that one of these results are expected after t=86 (e.g. after 86 days) for a vulnerability in category one. Hence, it is clear that once the "steady state" is achieved, for a vulnerability of category one, estimates of the probability of being exploited is 12.65% and the probability of being patched is 87.35%. Similarly, for vulnerabilities of categories two and three, the transition probability matrices can be obtained.

Transition probability matrices and resulting steady state vectors for those categories are given below. For the vulnerabilities in category two:

$$P = \begin{bmatrix} 0.8112 & 0.1888 & 0 & 0 & 0 \\ 0 & 0 & 0.081 & 0.119 & 0.8 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow$$

$$P^{(80)} = \begin{bmatrix} 0 & 0 & 01524 & 0 & 0.8476 \\ 0 & 0 & 0.1524 & 0 & 0.8476 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\overline{P^{(80)}} = \overline{P^{(0)}} P^{(80)} = [\,0 \quad 0 \quad 0.1524 \quad 0 \quad 0.8476\,].$$

For the vulnerabilities in category three:

$$P = \begin{bmatrix} 0.8196 & 0.1804 & 0 & 0 & 0 \\ 0 & 0 & 0.1476 & 0.0524 & 0.8 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow$$

$$P^{(84)} = \begin{bmatrix} 0 & 0 & 01790 & 0 & 0.821 \\ 0 & 0 & 0.1524 & 0 & 0.821 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\overline{P^{(84)}} = \overline{P^{(0)}} P^{(84)} = [\,0 \quad 0 \quad 0.1790 \quad 0 \quad 0.821\,].$$

Table 6 below summarizes the results. The number of iterations (steps) that it takes to reach the "steady states" and resulting row vectors of probabilities for each three categories of vulnerabilities are given in Table 3.

TABLE 6

Number of Iterations to Reach Steady State and Steady State Vector for each Category of Vulnerability

Category	Number of iterations	Steady state	Probability of being exploited	Probability of being patched	Sum
Low	86	[0.0000 0.0000 0.1265 0.0000 0.8735]	0.1265	0.8735	1
Medium	80	[0.0000 0.0000 0.1524 0.0000 0.8476]	0.1524	0.8476	1
High	84	[0.0000 0.0000 0.1790 0.0000 0.8210]	0.179	0.821	1

Having the steady state vector with the probabilities for patching and getting exploited, it is possible to calculate the risk of a particular vulnerability using the "risk factor". Previously, this risk factor was introduced as: Risk $v_i(t)$=Pr (v_i is in state 3 at time t)×Exploitability score (v_i).

The exploitability score for the vulnerability can be taken from the CVSS score as described earlier. With the results for the three different levels of vulnerabilities, a better index for the risk factor is available because the initial probabilities were not just chosen randomly but estimated using reliable data sources. As an example, a vulnerability in the lower level with an exploitability score of 2.4 can be investigated. For the risk factor of a vulnerability at t=50, for example, the Markov process can be used to determine with the resulting vector of the vulnerability that gives us the probabilities of being in each different state at that particular time. However, iterating a Markov process for each time would not be a very efficient process due to the analytical calculations. Therefore, it is possible to develop three different nonlinear statistical models that make it much more convenient for the designed calculation.

For example, consider a vulnerability given in the Table 7 below. With the published date and the exploitability score known for that vulnerability, the risk of being exploited can be calculated at a particular date from the published date. For the first vulnerability V1 (CVE 2016-0911), which is a low risk vulnerability, the risk factor is 0.2474. The other two categories of medium and high risk levels have vulnerabilities V2 (CVE 2016-2832) and V3 (CVE 2016-3230) and risk factors 0.3667 and 1.17702, respectively.

TABLE 7

Three Vulnerabilities in each Category with Details and Calculated Risk Factors

Vulnerability	Published date	CVSS score	Exploitability score	Age of the Vulnerability to the date Jun. 24, 2016 (t_j)	Risk factor ($R(v_i(t_i))$)
V_1 (CVE 2016-0911)	Jun. 19, 2016	1.9 (Low)	3.4	5	0.2474
V_2 (CVE 2016-2832)	Jun. 13, 2016	4.3 (medium)	2.8	11	0.3667
V_3 (CVE 2016-3230)	Jun. 15, 2016	9 (High)	8	9	1.1702

The risk factor can be graphed as a function of time. FIG. **9** shows the behavior of the risk factor of the middle level vulnerability V2 (CVE 2016-2832) over a time period of 101 days starting from Jun. 13, 2016. As shown in FIG. **9**, the risk factor increases rapidly within around first 10 days indicating that once a vulnerability is published, the risk of being exploited rapidly increases. Even after this rapid increase, the risk does not show a decreasing behavior. This specific behavior is due to the model structure of the vulnerability life cycle. That is, consisting with two absorbing states (being exploited and being patched), either one of two outcomes are possible for a given vulnerability. Thus, considering the state of being exploited as an absorbing state, the life cycle does not move to any other state beyond being exploited. This explains why the graph increases without decreasing over time.

Based on FIG. **9**, we can conclude that over time with a life cycle consisting two absorbing states, the risk factor of a given vulnerability increases rapidly and becomes stable at a higher level of risk without decreasing back. This behavior

exemplifies the threat any vulnerability would impose on an information system. As far as a proper patch is released and installed, a probable harm from a given vulnerability increases monotonously. However, it should not be misinterpreted in the view point that the risk from a given vulnerability never reduces. The absorbing Markovian model does not consider some of the interactions that might take place in the real world situations, but is developed to show the impact of a vulnerability over time.

Above, an analytical algorithm is developed that identifies a number of steps (in time) that the absorbing transition probability matrix of the vulnerability life cycle will reach a steady state. Thus, for a given vulnerability in the categories of low, medium, and high risk levels, the probability of being exploited and the probability of being patched can be calculated as a function of time. However, this process is time consuming and the Markovian iteration process might be quite difficult to perform. Using the approach described herein to find the minimum number of steps for each category, t=86 steps was identified for category one vulnerabilities in one example, t=80 steps was identified for category two vulnerabilities in the example, and t=84 steps was identified for category three vulnerabilities in the example. The probability of being exploited was then recorded at the each step. Thus, for each category, 2×86, 2×80, and 2×84 matrices of information are developed, respectively. An additional goal is to utilize this information and develop a statistical model for each category to be able to predict the probability of being exploited as a function of time and thus bypass at least some of the analytical difficulties.

A sample of the data for each category is shown in Table 8 below.

TABLE 8

Matrix Values used for Model Building under each Category

Low Vulnerability (0-3.9)		Medium Vulnerability (4-6.9)		High Vulnerability (7-10)	
Y_i	t_i	Y_i	t_i	Y_i	t_i
0.002897	1	0.0153	1	0.026618	1
0.024865	2	0.041188	2	0.054112	2
0.042929	3	0.062188	3	0.076645	3
0.057784	4	0.079223	4	0.095114	4
0.069998	5	0.093042	5	0.110251	5
0.080042	6	0.104252	6	0.122657	6
0.088302	7	0.113345	7	0.132825	7
0.095093	8	0.120722	8	0.141159	8
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
0.126521	77	0.152416	71	0.179021	75
0.126521	78	0.152416	72	0.179021	76
0.126521	79	0.152416	73	0.179021	77
0.126521	80	0.152416	74	0.179021	78
0.126521	81	0.152416	75	0.179021	79
0.126521	82	0.152416	76	0.179021	80
0.126521	83	0.152416	77	0.179021	81
0.126521	84	0.152416	78	0.179021	82
0.126521	85	0.152416	79	0.179021	83
0.126521	86	0.152416	80	0.179021	84

This data exhibits nonlinear behavior and thus multiple regression is not applicable. The model developer **142** is configured to develop one or more non-linear statistical models for each category. The general analytical focus of the statistical models are of the forms: Model 1: Y (exploitation probability)=

$$\alpha_0 + \alpha_1 \frac{1}{t} + \alpha_2 \ln t + \varepsilon$$

and Model 2: Y (exploitation probability)=

$$\beta_0 + \beta_1 \frac{1}{t} + \beta_2 \ln t + \varepsilon,$$

where Y is the probability of being exploited, α and β are the vector of coefficients or weights, t is the time given in steps, and ε is the modelling error. The method of maximum likelihood estimation was used to obtain the estimates of the coefficients that drives these models.

A good nonlinear statistical model developed by the model developer **142** for the low, medium, and high vulnerability categories is given below as Model 1, along with their R^2 (coefficient of determination), R_{adj}^2 (R^2 adjusted).

Low (Category one) risk vulnerabilities:

$$Y=0.084197-0.116756(1/t)-0.011321 \ln(t),$$

with $R^2=0.8684$, $R^2_{adj}=0.8653$.

Medium (Category two) risk vulnerabilities:

$$Y=0.111073-0.143992(1/t)+0.011461 \ln(t),$$

with $R^2=0.8888$, $R_{adj}^2=0.8859$.

High (Category three) risk vulnerabilities:

$$Y=0.133927-0.169314(1/t)+0.012375 \ln(t),$$

with $R^2=0.8988$, $R_{adj}^2=0.8963$. R^2 reflects on the quality of the proposed model.

By implementing another logarithmic filter to the initial model and homogenizing the variance of the data, the precision of Model 1 can be improved. A set of models was obtained that gives better results increasing the accuracy of the prediction approximately by 9% compared to Model 1. Model 2 equations for each of the categories are given below.

Low (Category one) risk vulnerabilities:

$$Y=0.135441-0.308532(1/t)+0.002030 \ln(\ln(t)),$$

with $R^2=0.9576$, $R_{adj}^2=0.9566$.

Medium (Category two) risk vulnerabilities:

$$Y=0.169518-0.356821(1/t)+0.007011 \ln(\ln(t)),$$

with $R^2=0.962$, R=0.961.

High (Category three) risk vulnerabilities:

$$Y=0.135441-0.308532(1/t)+0.002030 \ln(\ln(t)),$$

with $R^2=0.9588$, $R_{adj}^2=0.9577$.

Model 2 is a significant improvement in the R^2 over Model 1. Both models give very good predictions of the probability of exploitation as a function of time. However, Model 2 seems to give better predictions because of the additional logarithmic filtering applied to homogenize the variance further. Table 9 summarizes the 6 model equations with respective R^2 (coefficient of determination), R_{adj}^2 (R^2 adjusted) values for convenient comparison.

TABLE 9

Nonlinear Statistical Models to Estimate Probability of Being Exploited as a Function of Time

Category	Model Equation	R^2	R^2_{adj}
Model 1			
Low	Y = 0.084197 − 0.116756 (1/t) + 0.011321 ln(t)	0.8684	0.8653
Medium	Y = 0.111073 − 0.143992 (1/t) + 0.011461 ln(t)	0.8888	0.8859
High	Y = 0.133927 − 0.169314 (1/t) + 0.012375 ln(t)	0.8988	0.8963
Model 2			
Category	Model Equation	R^2	R^2_{adj}
Low	Y = 0.135441 − 0.308532 (1/t) − 0.002030 ln(ln t)	0.9576	0.9566
Medium	Y = 0.169518 − 0.356821 (1/t) − 0.007011 ln(ln t)	0.962	0.961
High	Y = 0.191701 − 0.383521 (1/t) − 0.00358 ln(ln t)	0.9588	0.9577

The R^2 (coefficient of determination), R_{adj}^2 (R^2 adjusted), and residual analysis were used using actual data to validate the accuracy and the quality of these models. R^2 is commonly used to measure the goodness of a statistical model and is defined as:

$$R^2 = \frac{SS_{Reg}}{SS_{Total}} = 1 - \frac{SS_{Reg}}{SS_{Total}},$$

where SS_{Res} or SSE is the Sum of Squares of Residual and SS_{Total} is the Total Sum of Squares. It is also referred to as the Coefficient of Determination. The R^2=0.96 states that the model is an excellent fit such that the 96% of the behavior in the response variable (probability of being exploited) is explained and predicted by the attributable variable (time−t) and only a 4% of the change in the response variable is not explained due to the variance.

In order to be more confident in interpreting the value of R^2, R^2_{adj} (R^2 adjusted) was also calculated to address the issue of bias. R^2_{adj} (R^2 adjusted) is defined by:

$$R^2_{adj} = 1 - \frac{(n-1)SS_{Reg}}{(n-p)SS_{Total}},$$

where n is the sample size and p is the number of risk factors (attributable variables) in our models. The closer the R^2 and R^2_{adj} to one, the higher the quality of the models.

A residual analysis of all the models was also performed to determine if the error factor has significantly contributed to the accuracy of the models. In all cases, the residual error was not significant. Finally, all the models were tested with actual data and the results were exceptional.

A best fitting of the three Statistical models is needed to calculate the "risk factor" conveniently. In other words, the need is to obtain a best fitting model that can replace the Markovian iteration and hence avoid the difficulty in estimating of the probabilities for time "t" earlier to the "steady state". These new models achieve that goal.

Thus, the previously-developed models can be further expanded upon and developed using the techniques described herein. The methods of calculating the initial probabilities and creating the transition probability matrix in using of the Markovian process was also improved. The CVSS data presented in CVE details website and calculated initial probabilities for discovering and exploiting a vulnerability based on the records on last 17 years data. Finally, two sets of three models were created for predicting the risk of a particular vulnerability being exploited as a function of time. The models presented were proven to have an excellent fit with the Markovian process probabilities. Thus, the Markovian process can be replaced using these models, since these models identify the steady states of being exploited or being patched for each vulnerability.

FIG. **10** illustrates a process for developing a non-linear model for predicting exploitability according to various examples described herein. The process flowchart in FIG. **10** can be viewed as depicting example steps performed by the computing environment **110**, although other computing systems and environments can perform the process. The flowchart in FIG. **10** provides merely one example of a functional sequence or arrangement of steps that can be employed to develop a model for predicting exploitability as described herein. Although the process is described in connection with the computing environment **110**, other computing environments, systems, and/or devices can perform the process. Additionally, although not explicitly stated below, among each of the process steps described, any number of intermediate data accessing, storing, and logging steps can be performed.

Turning to FIG. **10**, at step **1002**, the process can include the life cycle graph constructor **138** constructing a vulnerability life cycle graph as described herein. The vulnerability life cycle graph can include a number of state nodes each representative of a state of a vulnerability in a computing system as shown in FIG. **8**, for example. The life cycle graph can be constructed based on data stored in the network data **122**, for example, according to characteristics of one or a network of computing systems.

At step **1004**, the process can include the absorbing state probability matrix developer **140** determining initial probabilities of at least one state node among the state nodes in the life cycle graph constructed at step **1002**. Table 1, above, presents initial probabilities used in the example described herein. These initial probabilities can be determined based on data stored in the network data **122**, the risk metric data **124**, and/or other data. To estimate λ_1, for example, one may include any number of the total number of vulnerabilities in each category of a computing system, each ranging from 0 to 10 in magnitudes, along with information related to their discovery with respect to time. Similarly, for the other states, the number of vulnerabilities discovered, exploited before disclosed, exploited after discovery but before patched, patched before disclosure, and patched after disclosure can be relied upon under each CVSS score level.

The absorbing transition probability matrix developer **140** is configured to reference the CVSS scores available for each vulnerability for the discovered, patched, and exploited states. The absorbing transition probability matrix developer **140** can also reference the CVSS classifications available in the CVE detail website where available, and other information can be used in some cases. Examples of the other information include that provided in various vulnerability reports, such as the information given by the Stefan Frei and/or the Secunia Vulnerability information report, among others.

At step **1006**, the process can include the absorbing state probability matrix developer **140** developing an absorbing transition probability matrix based on the vulnerability life cycle graph and the initial probabilities determined at step

1004 as described above. The process can also include iterating the absorbing transition probability matrix over a number of cycles until the absorbing transition probability matrix reaches a steady state, where the number of cycles is representative of a period of time.

At step **1008**, the process can include the model developer **142** developing a non-linear model to determine a probability metric of the vulnerability being exploited or a probability metric of the vulnerability being patched based on the steady state information from step **1006**. Particularly, the steady state information from step **1006** can be used to develop a statistical model for each category of vulnerability to predict the probability of being exploited as a function of time. The steady state information can also be used to develop a statistical model for each category of vulnerability to predict the probability of being patched as a function of time. The general analytical form of the model or models can be similar to those described above. The models can include separate models developed by the model developer **142** for low, medium, and high vulnerability categories as described above.

FIG. **11** illustrates an example schematic block diagram of a computing device **1100** for the computing environment **110** shown in FIG. **2** according to various embodiments described herein. The computing device **1100** includes at least one processing system, for example, having a processor **1102** and a memory **1104**, both of which are electrically and communicatively coupled to a local interface **1106**. The local interface **1106** can be embodied as a data bus with an accompanying address/control bus or other addressing, control, and/or command lines.

In various embodiments, the memory **1104** stores data and software or executable-code components executable by the processor **1102**. For example, the memory **1104** can store executable-code components associated with the model engine **130** for execution by the processor **1102**. The memory **1104** can also store data such as that stored in the data store **120**, among other data.

It is noted that the memory **1104** can store other executable-code components for execution by the processor **1102**. For example, an operating system can be stored in the memory **1104** for execution by the processor **1102**. Where any component discussed herein is implemented in the form of software, any one of a number of programming languages can be employed such as, for example, C, C++, C#, Objective C, JAVA®, JAVASCRIPT®, Perl, PHP, VISUAL BASIC®, PYTHON®, RUBY, FLASH®, or other programming languages.

As discussed above, in various embodiments, the memory **1104** stores software for execution by the processor **1102**. In this respect, the terms "executable" or "for execution" refer to software forms that can ultimately be run or executed by the processor **1102**, whether in source, object, machine, or other form. Examples of executable programs include, for example, a compiled program that can be translated into a machine code format and loaded into a random access portion of the memory **1104** and executed by the processor **1102**, source code that can be expressed in an object code format and loaded into a random access portion of the memory **1104** and executed by the processor **1102**, or source code that can be interpreted by another executable program to generate instructions in a random access portion of the memory **1104** and executed by the processor **802**, etc.

An executable program can be stored in any portion or component of the memory **1104** including, for example, a random access memory (RAM), read-only memory (ROM), magnetic or other hard disk drive, solid-state, semiconductor, universal serial bus (USB) flash drive, memory card, optical disc (e.g., compact disc (CD) or digital versatile disc (DVD)), floppy disk, magnetic tape, or other types of memory devices.

In various embodiments, the memory **1104** can include both volatile and nonvolatile memory and data storage components. Volatile components are those that do not retain data values upon loss of power. Nonvolatile components are those that retain data upon a loss of power. Thus, the memory **1104** can include, for example, a RAM, ROM, magnetic or other hard disk drive, solid-state, semiconductor, or similar drive, USB flash drive, memory card accessed via a memory card reader, floppy disk accessed via an associated floppy disk drive, optical disc accessed via an optical disc drive, magnetic tape accessed via an appropriate tape drive, and/or other memory component, or any combination thereof. In addition, the RAM can include, for example, a static random access memory (SRAM), dynamic random access memory (DRAM), or magnetic random access memory (MRAM), and/or other similar memory device. The ROM can include, for example, a programmable read-only memory (PROM), erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), or other similar memory device.

The processor **1102** can be embodied as one or more processors **1102** and the memory **1104** can be embodied as one or more memories **804** that operate in parallel, respectively, or in combination. Thus, the local interface **1106** facilitates communication between any two of the multiple processors **1102**, between any processor **1102** and any of the memories **1104**, or between any two of the memories **1104**, etc. The local interface **1106** can include additional systems designed to coordinate this communication, including, for example, a load balancer that performs load balancing.

As discussed above, model engine **130** can be embodied, at least in part, by software or executable-code components for execution by general purpose hardware. Alternatively the same can be embodied in dedicated hardware or a combination of software, general, specific, and/or dedicated purpose hardware. If embodied in such hardware, each can be implemented as a circuit or state machine, for example, that employs any one of or a combination of a number of technologies. These technologies can include, but are not limited to, discrete logic circuits having logic gates for implementing various logic functions upon an application of one or more data signals, application specific integrated circuits (ASICs) having appropriate logic gates, field-programmable gate arrays (FPGAs), or other components, etc.

The flowchart or process diagrams in FIGS. 7A, **7B**, and **10** are representative of certain processes, functionality, and operations of the embodiments discussed herein. Each block can represent one or a combination of steps or executions in a process. Alternatively or additionally, each block can represent a module, segment, or portion of code that includes program instructions to implement the specified logical function(s). The program instructions can be embodied in the form of source code that includes human-readable statements written in a programming language or machine code that includes numerical instructions recognizable by a suitable execution system such as the processor **1102**. The machine code can be converted from the source code, etc. Further, each block can represent, or be connected with, a circuit or a number of interconnected circuits to implement a certain logical function or process step.

Although the flowchart or process diagrams in FIGS. **7A**, **7B**, and **10** illustrate a specific order, it is understood that the

order can differ from that which is depicted. For example, an order of execution of two or more blocks can be scrambled relative to the order shown. Also, two or more blocks shown in succession can be executed concurrently or with partial concurrence. Further, in some embodiments, one or more of the blocks can be skipped or omitted. In addition, any number of counters, state variables, warning semaphores, or messages might be added to the logical flow described herein, for purposes of enhanced utility, accounting, performance measurement, or providing troubleshooting aids, etc. Such variations, as understood for implementing the process consistent with the concepts described herein, are within the scope of the embodiments.

Also, any logic or application described herein, including the model engine **130** that are embodied, at least in part, by software or executable-code components, can be embodied or stored in any tangible or non-transitory computer-readable medium or device for execution by an instruction execution system such as a general purpose processor. In this sense, the logic can be embodied as, for example, software or executable-code components that can be fetched from the computer-readable medium and executed by the instruction execution system. Thus, the instruction execution system can be directed by execution of the instructions to perform certain processes such as those illustrated in FIGS. **7**A, **7**B, and **10**. In the context of the present disclosure, a non-transitory computer-readable medium can be any tangible medium that can contain, store, or maintain any logic, application, software, or executable-code component described herein for use by or in connection with an instruction execution system.

The computer-readable medium can include any physical media such as, for example, magnetic, optical, or semiconductor media. More specific examples of suitable computer-readable media include, but are not limited to, magnetic tapes, magnetic floppy diskettes, magnetic hard drives, memory cards, solid-state drives, USB flash drives, or optical discs. Also, the computer-readable medium can include a RAM including, for example, an SRAM, DRAM, or MRAM. In addition, the computer-readable medium can include a ROM, a PROM, an EPROM, an EEPROM, or other similar memory device.

Disjunctive language, such as the phrase "at least one of X, Y, or Z," unless specifically stated otherwise, is to be understood with the context as used in general to present that an item, term, etc., can be either X, Y, or Z, or any combination thereof (e.g., X, Y, and/or Z). Thus, such disjunctive language is not generally intended to, and should not, imply that certain embodiments require at least one of X, at least one of Y, or at least one of Z to be each present.

It should be emphasized that the above-described embodiments of the present disclosure are merely possible examples of implementations set forth for a clear understanding of the principles of the disclosure. Many variations and modifications can be made to the above-described embodiment(s) without departing substantially from the spirit and principles of the disclosure. All such modifications and variations are intended to be included herein within the scope of this disclosure and protected by the following claims.

At least the following is claimed:

1. A method to develop a model for predicting exploitability, comprising:

- constructing a vulnerability life cycle graph, the vulnerability life cycle graph including a plurality of state nodes each representative of a state of a vulnerability in a computing system;
- determining an initial probability of at least one state node among the plurality of state nodes;
- developing an absorbing transition probability matrix based on the vulnerability life cycle graph and the initial probability of the at least one state;
- iterating the absorbing transition probability matrix over a number of cycles until the absorbing transition probability matrix reaches a steady state; and
- developing a non-linear model to determine a probability metric of the vulnerability being exploited or a probability metric of the vulnerability being patched based on the steady state.

2. The method according to claim **1**, wherein the number of cycles is representative of a period of time.

3. The method according to claim **1**, wherein the iterating provides a probability metric of the vulnerability being exploited as a function of time and a probability metric of the vulnerability being patched as a function of time.

4. The method according to claim **1**, wherein determining the initial probability of the at least one state node comprises calculating a probability estimate for the at least one state node for three categories of the Common Vulnerability Scoring System (CVSS) framework.

5. The method according to claim **1**, wherein the plurality of state nodes include at least one absorbing state and, during the iterating, the initial probability is absorbed into the least one absorbing state.

6. The method according to claim **5**, wherein the at least one absorbing state comprises at least one of a patched state or an exploited state.

7. The method according to claim **1**, wherein determining the initial probability of the at least one state node comprises calculating a probability estimate for the at least one state node as a proportion of a cumulative number of vulnerabilities in the computing system over a number of years.

8. The method according to claim **1**, wherein determining the initial probability of the at least one state node comprises identifying a probability estimate for the at least one state node from a reference dataset.

9. The method according to claim **1**, further comprising calculating at least one of the probability metric of the vulnerability being exploited or the probability metric of the vulnerability being patched based on the non-linear model.

10. The method according to claim **9**, further comprising calculating a timeframe associated with at least one of the probability metric of the vulnerability being exploited or the probability metric of the vulnerability being patched.

11. The method according to claim **10**, further comprising communicating the timeframe and at least one of the probability metric of the vulnerability being exploited or the probability metric of the vulnerability being patched to an information technology specialist to take remedial measures.

12. A system to develop a model for predicting exploitability, comprising:

- a memory device configured to store computer-readable instructions thereon; and
- at least one processing device directed, through execution of the computer-readable instructions, to:
  - construct a vulnerability life cycle graph, the vulnerability life cycle graph including a plurality of state nodes each representative of a state of a vulnerability in a computing system;
  - determine an initial probability of at least one state node among the plurality of state nodes;
  - develop an absorbing transition probability matrix based on the vulnerability life cycle graph and the initial probability of the at least one state;

iterate the absorbing transition probability matrix over a number of cycles until the absorbing transition probability matrix reaches a steady state; and

develop a non-linear model to determine a probability metric of the vulnerability being exploited or a probability metric of the vulnerability being patched based on the steady state.

13. The system according to claim **12**, wherein the number of cycles is representative of a period of time.

14. The system according to claim **12**, wherein the at least one processing device is further configured to iterate the absorbing transition probability matrix to provide a probability metric of the vulnerability being exploited as a function of time and a probability metric of the vulnerability being patched as a function of time.

15. The system according to claim **12**, wherein the plurality of state nodes include at least one of a patched state or an exploited state.

16. A method to develop a model for predicting exploitability, comprising:

constructing a vulnerability life cycle graph, the vulnerability life cycle graph including a plurality of state nodes each representative of a state of a vulnerability in a computing system;

determining an initial probability of at least one state node among the plurality of state nodes;

developing an absorbing transition probability matrix based on the vulnerability life cycle graph and the initial probability of the at least one state;

iterating the absorbing transition probability matrix over a number of cycles until the absorbing transition probability matrix reaches a steady state; and

developing a non-linear model to determine a probability metric of the vulnerability being exploited or a probability metric of the vulnerability being patched based on the steady state.

17. The method according to claim **16**, wherein the iterating provides a probability metric of the vulnerability being exploited as a function of time and a probability metric of the vulnerability being patched as a function of time.

18. The method according to claim **16**, wherein determining the initial probability of the at least one state node comprises calculating a probability estimate for the at least one state node for three categories of the Common Vulnerability Scoring System (CVSS) framework.

* * * * *