(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
01 June 2023 (01.06.2023)

WIPO | PCT

(10) International Publication Number
WO 2023/094931 A1

(51) International Patent Classification:
G06F 21/57 (2013.01) *H04L 9/40* (2022.01)

(21) International Application Number:
PCT/IB2022/060940

(22) International Filing Date:
14 November 2022 (14.11.2022)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

63/264,550	24 November 2021 (24.11.2021)	US
63/283,376	26 November 2021 (26.11.2021)	US
63/283,378	26 November 2021 (26.11.2021)	US
63/283,379	26 November 2021 (26.11.2021)	US

(71) Applicant: **WIZ, INC.** [US/US]; One Manhattan West, 57th Floor, New York, New York 10001 (US).

(72) Inventors: **HERZBERG, Raaz**; c/o Wiz, Inc., One Manhattan West, 57th Floor, New York, New York 10001 (US). **OLIVER, Yaniv Joseph**; c/o Wiz, Inc., One Manhattan West, 57th Floor, New York, New York 10001 (US). **HAZAN, Osher**; c/o Wiz, Inc., One Manhattan West, 57th Floor, New York, New York 10001 (US). **BEN DAVID, Niv Roit**; c/o Wiz, Inc., One Manhattan West, 57th Floor, New York, New York 10001 (US). **LUTTWAK, Ami**; c/o Wiz, Inc., One Manhattan West, 57th Floor, New York, New York 10001 (US). **REZNIK, Roy**; c/o Wiz, Inc., One Manhattan West, 57th Floor, New York, New York 10001 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*

WO 2023/094931 A1

(54) Title: DETECTING VULNERABILITIES IN CONFIGURATION CODE OF A CLOUD ENVIRONMENT UTILIZING INFRASTRUCTURE AS CODE

(57) **Abstract:** A system and method for detecting a vulnerable code object in configuration code for deploying instances in a cloud computing environment is disclosed. The method includes accessing a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance; querying a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and generating a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

DETECTING VULNERABILITIES IN CONFIGURATION CODE OF A CLOUD ENVIRONMENT UTILIZING INFRASTRUCTURE AS CODE

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 63/264,550 filed on November 24, 2021. This application also claims the benefit of U.S. Provisional Application No. 63/283,376 filed on November 26, 2021, U.S. Provisional Application No. 63/283,378 filed on November 26, 2021, and U.S. Provisional Application No. 63/283,379 filed on November 26, 2021, the contents of which are hereby incorporated by reference. All of the applications referenced above are hereby incorporated by reference.

TECHNICAL FIELD

[0002] The present disclosure relates generally to cybersecurity and, in particular, to improved scanning of virtual instances utilizing infrastructure as code.

BACKGROUND

[0003] As users migrate data storage, processing, and management tasks to decentralized, off-location devices, platforms, and services, the limitations of such devices, platforms, and services, also referred to as cloud environments, platforms, and the like, may impact a user's data operations. Specifically, vulnerabilities within cloud-deployed resources and processes may present unique challenges requiring remediation. Due to the scale and structure of cloud systems, detection of workload vulnerabilities, which detection may be readily-provided in non-cloud deployments, may require numerous, complex tools and operations.

[0004] Current solutions to cloud workload vulnerability scanning challenges require the deployment of specialized tools, including scanning agents directed to maintenance of virtual machines (VMs), where operation and maintenance of such tools may be costly, time-consuming, or both. Agent-dependent processes fail to provide for scanning of containers, such as containers managed using Kubernetes®, and other, like, container-management platforms, and may fail to provide for coverage of serverless applications. Where such agent-implementation processes fail to provide for full cloud workload

vulnerability scanning, additional methods, such as snapshot-based scanning, may supplement implemented solutions.

[0005] Snapshot-based scanning, wherein static "snapshots" of processes, services, data, and the like, are analyzed in an environment separate from the source environment, provides for agentless scanning. Snapshot-based scanning is applied in various fields, including computer forensics, to provide for analysis of services, processes, data, and the like, in locations or environments other than those from which the snapshots are collected, as well as retrospective analysis. However, the applicability of snapshot-based scanning is limited in multi-tenant systems, such as shared cloud platforms, as cloud tenants may desire high levels of data protection during snapshot generation, transfer, and analysis. Further, snapshot-based scanning methods, as well as hybrid methods including both agent-implemented and snapshot-based methods, may be inapplicable to certain cloud system structures and environments, which may include various objects, processes, and the like, which such methods may not be configured to process, as such processing may require, as examples, separate analysis of container repositories, VM snapshots, and application programming interfaces (API) for serverless applications, where existing solutions fail to provide such integrated functionality.

[0006] Further complicating matters is deployment of cloud environments utilizing infrastructure as code (IaC) systems. While aimed at decreasing human error when deploying cloud environments, there is often a drift from the original configuration code to the current state of the production environment. A complication may arise due, for example, to different teams working on the development environment (configuration code) and the production environment (deployed instances). Current tools such as Checkov and Accurics allow to scan for misconfigurations and policy violations, but are limited to scanning only configuration code. CI/CD (continuous integration / continuous deployment) and drifting configurations mean that scanning the configuration code is not always enough to get a precise understanding of where threats and vulnerabilities currently exist, since this is a moving target.

[0007] It is apparent that it would be advantageous to provide a solution which can scan for vulnerabilities in an improved and efficient manner.

[0008] Furthermore, it would, therefore, be advantageous to provide a solution that would overcome the challenges noted above.

SUMMARY

[0009] A summary of several example embodiments of the disclosure follows. This summary is provided for the convenience of the reader to provide a basic understanding of such embodiments and does not wholly define the breadth of the disclosure. This summary is not an extensive overview of all contemplated embodiments, and is intended to neither identify key or critical elements of all embodiments nor to delineate the scope of any or all aspects. Its sole purpose is to present some concepts of one or more embodiments in a simplified form as a prelude to the more detailed description that is presented later. For convenience, the term “some embodiments” or “certain embodiments” may be used herein to refer to a single embodiment or multiple embodiments of the disclosure.

[0010] Certain embodiments disclosed herein include a method for detecting a vulnerable code object in configuration code for deploying instances in a cloud computing environment. The method comprises: accessing a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance; querying a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and generating a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

[0011] Certain embodiments disclosed herein also include a non-transitory computer readable medium having stored thereon causing a processing circuitry to execute a process, the process comprising: accessing a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance; querying a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and generating a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

[0012] Certain embodiments disclosed herein also include a system for detecting a vulnerable code object in configuration code for deploying instances in a cloud computing environment. The system comprises: a processing circuitry; and a memory, the memory containing instructions that, when executed by the processing circuitry, configure the system to: access a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance; query a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and generate a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The subject matter disclosed herein is particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other objects, features, and advantages of the disclosed embodiments will be apparent from the following detailed description taken in conjunction with the accompanying drawings.

[0014] Fig. 1 is a network diagram of a monitored cloud computing environment utilizing infrastructure as code (IaC) utilized to describe the various embodiments.

[0015] Fig. 2 is a flowchart of a method for inspecting configuration code utilizing a security graph, implemented in accordance with an embodiment.

[0016] Fig. 3 is a schematic illustration of a portion of a security graph for cybersecurity risk assessment of virtual instances in a cloud computing environment, implemented in accordance with an embodiment.

[0017] Fig. 4 is a schematic illustration of a code inspector implemented according to an embodiment.

[0018] Fig. 5 is a code object, shown in accordance with an embodiment.

DETAILED DESCRIPTION

[0019] It is important to note that the embodiments disclosed herein are only examples of the many advantageous uses of the innovative teachings herein. In general, statements

made in the specification of the present application do not necessarily limit any of the various claimed embodiments. Moreover, some statements may apply to some inventive features but not to others. In general, unless otherwise indicated, singular elements may be in plural and vice versa with no loss of generality. In the drawings, like numerals refer to like parts through several views.

[0020] Infrastructure as code (IaC) allows fast and reliable deployment of workloads and accounts in cloud based computing environments. A workload may be, for example, a virtual machine, a container, or a serverless function. A virtual machine may be implemented for example as an Oracle® VM VirtualBox hypervisor, a container may be implemented on a Kubernetes® platform, and serverless function may be implemented as Amazon® Web Services (AWS) Lambda. Accounts may be user accounts, service accounts, roles, and the like.

[0021] Typically, the deployed environment, also known as a production environment, differs over time from the initial deployment configuration, due for example to upgrades and patches implemented in production but not always updated in configuration code. This can occur for example due to human error. Furthermore, many deployment environments utilize a continuous integration, continuous deployment (CI/CD) approach, for which a plurality of deployment environments are used. A cloud computing environment is designed, in an embodiment, utilizing infrastructure as code tool and other development tools in a development (i.e., dev) environment, and deployed to a test environment where code is tested. In an embodiment, code which passes tests, benchmarks, and the like, is then deployed to a staging environment which is similar to the production environment. In each of these environments, a single code object can correspond to multiple machines which are deployed based on the code object, each of which can be host to cybersecurity vulnerabilities. For example, a code object includes a code instruction which when executed to deploy a workload, configures a workload having a misconfiguration. If the misconfiguration is detected and corrected in the production environment, the code remains faulty and therefore the next machine deployed based off of the code will also have a misconfiguration.

[0022] In an embodiment, a security graph includes a representation of a production environment. The security graph is utilized in inspecting the configuration code to

ascertain that code objects comply with the specification of the production environment. For example, the security graph is queried, in an embodiment, to detect a node representing a workload, which corresponds to an identifier of a code object. By utilizing information represented in the security graph of the production environment and applying it to code objects, cybersecurity threats can be detected faster, and to the point where they originate.

[0023] Alerts may be generated to indicate that the configuration code would result in a new production environment which is deficient, for example due to vulnerability, when compared with the current production environment.

[0024] While declaratory code is used precisely because it is intuitive for humans to read and write declaratory code, it should be appreciated that inspecting such code for cybersecurity issues is not a task that can be performed by humans. Specifically, inspecting code to detect a cybersecurity issue needs to be performed in a reliable and consistent manner, and done so repeatedly over often thousands of lines of code. Even if it were practical for a human to read through thousands of lines of computer code within any meaningful time frame (cloud computing environments are elastic and constantly changing) , doing so while searching for hundreds of thousands of various cybersecurity issues is impossible. Furthermore, humans are not capable of performing such tasks repeatedly and reliably, as they apply objective standards to what is a cybersecurity issue.

[0025] By contrast, an embodiment of the system disclosed herein applies objective criteria in detection of cybersecurity issues, and does so in a manner which is reliable, consistent, and in a timeframe which is relevant to the operation of a cloud computing environment. Additionally, methods disclosed herein provide for improved efficiency of computer systems, by reducing use of memory, processors, and the like.

[0026] Fig. 1 is a network diagram 100 of a monitored cloud computing environment utilizing infrastructure as code (IaC) utilized to describe the various embodiments.

[0027] A client device 110 generates a configuration code file 120 based on input from one or more users (e.g., software programmers). In an embodiment, a client device is a personal computer, a tablet, a laptop, and the like. In some embodiment, a client device 110 is used to access a server (not shown) which provides a computing environment into which input can be provided. It should be apparent that the client device 110 is shown

here for simplicity and pedagogical purposes, and that the configuration code file 120 is generated, in other embodiments, by the client device, a virtual workload in a cloud computing environment, a combination thereof, and the like. In certain embodiments, the configuration code file 120 is generated by multiple different client devices. For example, a plurality of users may each utilize a different client device and update a single configuration code file 120, for example, with code objects. In some embodiments, a single client device 110 generates multiple configuration code files.

[0028] In an embodiment the configuration code file 120 is implemented in a declaratory computer language. In a declaratory computer language, a user declares resources they would like to have as code objects, and an orchestrator, such as orchestrator 130, is configured to deploy workloads in a cloud computing environment based on the declarations. For example, an orchestrator 130 is configured, in an embodiment, to translate a declaratory code to a configuration code, which includes instructions which when executed configure a cloud computing environment to deploy a workload, virtual instance, and the like.

[0029] In certain embodiments, multiple configuration code files 120 may be utilized. For example, a user may operate multiple cloud environments, each with its own configuration code. For example, a first configuration code file is directed to deploying a cloud computing environment over Microsoft® Azure, while a second configuration code file is directed to deploying a cloud computing environment over Amazon® Web Services (AWS).

[0030] As another example, a user can declare a first resource type (e.g., virtual machine) for a first cloud environment (e.g., AWS) and for a second cloud environment (Google® Cloud Platform – GCP) in a first configuration code file, and a second resource type (e.g., software container) for the first cloud environment (AWS) and the second cloud environment (GCP) in a second configuration code file.

[0031] In an embodiment, an orchestrator 130 is configured to receive the configuration code file 120. In certain embodiments, the orchestrator 130 is configured to initiate actions in a cloud computing environment 140, for example, to deploy workloads, instances, user accounts, service accounts, combinations thereof, and the like, based on declarations of the configuration code file 120. In an embodiment, an instance is a virtual instance, and

may be, for example a virtual machine 142, software container 144, a serverless function 146, and the like.

[0032] In some embodiments, the orchestrator 130 is configured to deploy workloads by assigning (also known as provisioning) cloud computing environment resources, such as processors, memory, storage, etc. to the workload. In an embodiment, workloads are deployed in a production environment, which is a cloud computing environment having operable code, used for providing access to data and providing software services. In some embodiments, configuration code is implemented in a development (dev) environment, which also utilizes a cloud computing environment.

[0033] In some embodiments, a plurality of workloads are associated with a first code object (not shown) of the configuration code file 120. Workloads which are all deployed based on a same code object (i.e., the first code object) are known as a virtual instance (or "instance") of the first code object. In an embodiment, associating a workload with a code object includes assigning a name to the instance based on an identifier of the code object.

[0034] This provides an advantage where it is required to deploy multiple instances which share similar configurations, such as web servers providing access to a website. Rather than configure each instance manually and individually, an orchestrator 130 is configured to deploy a number of the same workload based on the configuration code file 120.

[0035] In some embodiments the orchestrator 130 may configure a cloud-native orchestrator (not shown) in the cloud computing environment 140 to deploy the instances. This may be advantageous, for example, where instances need to be deployed in different cloud environments.

[0036] For example, the same instances may be deployed simultaneously on Google® Cloud Platform (GCP), Amazon® Web Services (AWS), or Microsoft® Azure. This can be achieved by configuring the orchestrator 130 to generate native instructions for a cloud native orchestrator in each environment to deploy such instances. The native instructions are generated by the orchestrator 130 in an embodiment. The instructions are generated based on objects detected in the configuration code file 120.

[0037] This method of deploying instances decreases errors by eliminating the need for a user to manually deploy each instance and configure each instance separately, and is also thus a faster method of deployment. A human is not able to consistently and reliably

initiate deployment of virtual instances, and then configure hundreds or thousands of such instances to match the same specification. In the example above a first load balancer may be deployed in a first cloud computing environment, and a second load balancer may be deployed in a second cloud computing environment, each cloud computing environment having different infrastructure from each other, wherein the first load balancer and the second load balancer are deployed based on the same code object from a configuration code file.

[0038] In an embodiment, the first cloud computing environment 140 is coupled with a second cloud computing environment 150, which is configured to inspect the first cloud computing environment 140 for cybersecurity threats. In an embodiment, the second cloud computing environment 150 (also referred to as inspection environment 150) is further configured to receive the configuration code file 120.

[0039] In some embodiments, the second cloud environment 150 is utilized for inspecting the first cloud computing environment 140 and generating cybersecurity risk assessments for instances deployed in the first cloud computing environment 140.

[0040] In certain embodiments, the second cloud environment 150 includes a plurality of inspectors, such as inspector 160. An inspector is a workload which is configured to inspect another workload for cybersecurity objects, such as a secret, a file, a folder, a registry value, a weak password, a certificate, a malware object, a hash, a misconfiguration, a vulnerability, an exposure, a combination thereof, and the like. In an embodiment, an inspector 180 is configured to inspect for a plurality of cybersecurity object types.

[0041] For example, in an embodiment, an inspector is configured to inspect the virtual machine 142 for a predetermined cybersecurity object, in response to receiving an instruction to inspect the virtual machine 142. In an embodiment the instruction is received through an API (not shown) of the first cloud computing environment 140. In some embodiments, an inspectable disk is generated based on a volume (not shown) attached to the virtual machine 142, and the inspectable disk is provided to the second cloud computing environment 150 for inspection. In an embodiment, generating an inspectable disk includes generating a clone of the volume, generating a copy of the volume, generating a snapshot of the volume, and the like. In an embodiment, a software container

is deployed in the second cloud computing environment 150 and attached to a volume generated in the second cloud computing environment 150 based on the received snapshot. The inspector 160 is configured, in an embodiment, to inspect the attached volume for a predefined cybersecurity object type. In an embodiment, the inspector 160 is configured to generate data which is stored on a security graph 170. In some embodiments, a node is stored on the security graph 170 to represent an inspected resource. In an embodiment, data generated by the inspector 160 is stored on the node representing the workload which the inspector 160 inspected for a cybersecurity object.

[0042] In an embodiment, the security graph 170 is stored on a graph database. The security graph 170 includes a representation of a cloud computing environment. In an embodiment, the representation includes a plurality of nodes, at least a portion of which each represent a resource or a principal. A resource is a cloud entity which provides access to a service, computer hardware (e.g., processor, memory, storage, and the like), and the like. In an embodiment, a resource is a workload, such as a virtual machine, serverless function, software container, and the like. A principal is a cloud entity which is authorized to initiate actions in a cloud computing environment, and is authorized to act on a resource. In an embodiment, a principal is a user account, a user group, a service account, and the like.

[0043] In certain embodiments, the security graph 170 further includes enrichment nodes, which represent certain redetermined functionalities, network access, and the like. For example, an enrichment node may be used to represent access to a public network, such as the Internet. Thus, a node representing a workload which has access to a public network, or can be accessed through a public network, is connected in the security graph 170 to an enrichment node representing public network access.

[0044] In an embodiment, a code inspector 180 is further deployed in the second cloud computing environment 150. In some embodiments, a plurality of code inspectors are deployed. In certain embodiments, configuration code is generated by multiple different type of platforms, such as Pulumi®, Terraform®, and the like.

[0045] In some embodiments, a first code inspector is configured to inspect configuration code generated using Pulumi®, while a second code inspector is configured to inspect configuration code generated using Terraform®. In an embodiment, the code inspector

180 is realized as a workload, such as an application deployed on a software container, configured to receive configuration code and inspect the configuration code to detect a predetermined type of code object. In an embodiment, a type of code object is, for example, a secret (such as a public key, or a private key), a resource type, an application identifier, a policy identifier, a role identifier, a status of a flag, and the like. A flag status indicates, in an embodiment, that a certain object is allowed to perform certain actions, such as network access, or assume a role, such as an administrator role (in the case of a user or service account).

[0046] In an embodiment, the code inspector 180 is configured to match the detected object to a node in the security graph 170. This is discussed in more detail with respect to Fig. 2 below.

[0047] Fig. 2 is an example flowchart 200 of a method for inspecting configuration code utilizing a security graph, implemented in accordance with an embodiment. In an embodiment, configuration code in a development (dev) environment is inspected based on a security graph which is generated at least in part based on a production environment.

[0048] A production environment is rarely, if at all, identical to the environment which is deployed initially by code. This is due to, for example, upgrades and patches implemented in the production environment to address issues caused by the code deployment. Drifting configuration, or configuration drift, describes how a production environment, over time, 'drifts' further away from the initial configuration code design. Therefore, inspecting only one environment for cybersecurity threats is not enough, and it is advantageous to inspect both.

[0049] In an embodiment, the security graph includes representations of the configuration code (e.g., representing code objects) and the production environment (e.g., representing resources and principals). By inspecting a configuration code file based on a security graph generated from data of a production environment, insight can be gained, and deployment issues may be caught early on, for example to identify instances which if deployed based on a current version of configuration code would include a version of software which the production environment has already upgraded to a newer version. In an embodiment, the method is performed by a configuration code inspector, such as the code inspector 180.

[0050] At S210, configuration code is received. In an embodiment, the configuration code includes a plurality of code objects. In certain embodiments, a portion of the code objects correspond to instances which are deployed in a cloud computing environment. In an embodiment, the configuration code is scanned or otherwise inspected as a textual object. For example, a configuration code is searched for regular expressions (regex), strings, and the like.

[0051] At S220, a first code object is extracted from the received code. Extracting a code object includes, in an embodiment, searching the text of a configuration code file for a predetermined string. For example, a code object may be a text field identifying a type of workload, a name of a workload, a network address, a name in a namespace, a role, a permission, and the like. In some embodiments, a plurality of code objects are extracted from the received code.

[0052] At S230, a security graph is traversed to detect a node in the graph corresponding to the extracted first code object. In an embodiment, traversing the security graph includes sending a request through an API of a graph database hosting the security graph to search the graph for a string, a value, and the like, which corresponds to the first code object. For example, if the first code object includes a secret, such as a private key (i.e., an alphanumerical representation), the security graph is traversed to detect a node which represents a matching public key (e.g., public key node). In an embodiment, the public key node is connected to a resource node representing a resource which utilizes the public key.

[0053] In some embodiments, a query directed at the security graph includes a plurality of clauses. In an embodiment, multiple-clause query is generated to search for container nodes (i.e., nodes representing containers) which are connected to a node representing the public key. It is noted that detecting a node which corresponds to the extracted first object includes, in an embodiment, detecting a node which is not a node representing a workload corresponding to the first object.

[0054] For example, executing code of the first code object results, in an embodiment, in deploying a first load balancer in a virtual private cloud (VPC). In an embodiment, a node is generated in a security graph to represent the first load balancer deployed in a cloud

computing environment. The node representing the load balancer is connected to a node representing the VPC.

[0055] An advantage of the disclosed method is that attributes of the first code object detected in the graph allows detecting nodes representing cybersecurity issues, nodes representing workloads, enrichment nodes, and the like, prior to the generation of an instance based on the code object. This allows detecting a security risk in an instance prior to it being deployed in a computing environment. In the above example, as the code of the first code object includes instructions to deploy in the VPC, the VPC node is detected (based, for example, on detecting an identifier of the VPC in the code) in the security graph. Cybersecurity risks represented by nodes connected to the VPC node are detected, for example by querying the security graph.

[0056] At S240, a check is performed to determine if a node is detected. If 'no' execution may continue at S270. In an embodiment, if a node is not detected (e.g., the node does not exist), a new node is generated in the security graph to represent the first code object. If a node is detected execution continues to S250.

[0057] At S250, a check is performed to determine if the detected node corresponds to a previously determined cybersecurity issue, such as a cybersecurity risk factor, vulnerability, misconfiguration, and the like. A risk factor, vulnerability, misconfiguration, and the like, may be, for example, access to a network resource (such as the internet), access from a network resource, outdated software, privilege escalation, and the like. In an embodiment, a risk factor score is further determined. In some embodiments, the score indicates the severity of the risk, such as 'low', 'medium', 'high', and 'critical'. In an embodiment, the previously determined cybersecurity issue is detected by inspecting a disk for a cybersecurity object. In some embodiments, a detected cybersecurity issue is represented as a node in a security graph, connected to a node representing a resource on which the cybersecurity issue was detected.

[0058] In an embodiment, a mitigation instruction corresponding to the risk factor score is executed. In some embodiments, the risk factor is indicated by metadata associated with the detected node of S240. If the detected node corresponds to a previously determined cybersecurity issue execution continues at S260; otherwise, execution continues at S270.

[0059] In an embodiment, a vulnerability is represented on the security graph by a node. As an example, a node representing a workload is connected to a node representing a vulnerability. Where a workload node is the detected node, a cybersecurity vulnerability is associated with the code object.

[0060] At optional S260 a notification is generated to indicate that a security risk has been detected in the configuration code. In an embodiment the notification is sent to a client device, a user account, a combination thereof, and the like, which authored the code. Code authors are determined, in an embodiment, by a user account identifier present in the configuration code.

[0061] In some embodiments, the notification includes an indicator to specify why the notification is generated. In certain embodiments an instruction to perform a mitigation action is generated. In the example above, an alert (i.e., notification) is generated in response to detecting that a workload includes an outdated software version, and the alert includes the current software version which would need to be configured in the configuration code in order to mitigate the risk of deploying a workload with an outdated software version.

[0062] At S270 a check is performed to determine if another code object should be inspected. If 'yes' execution continues at S220, otherwise execution terminates.

[0063] Fig. 3 is a schematic illustration of a portion of a security graph 300 for cybersecurity risk assessment of virtual instances in a cloud computing environment, implemented in accordance with an embodiment. The graph 300, which in an embodiment is stored in a graph database, includes a plurality of nodes. In an embodiment, a node represents a resource, principal, metadata, enrichment data, and the like.

[0064] In an embodiment, the graph 300 includes a first cloud key node 310 (representing a first cloud key) and a second cloud key node 320 (representing a second cloud key), which are connected to a user account node 340 (representing a user account). A third cloud key node 330 (representing a third cloud key) is connected to a service account node 360 (representing a service account). The user account node 340 and service account node 360 are connected to an identity and access management (IAM) object node 350 (representing an IAM object).

[0065] In an embodiment, a cloud key provides temporary access, permanent access, and the like, between a first workload and a second workload. In some embodiments, one or more first workloads and one or more second workloads may be on the same tenant, on different tenants, or on a combination thereof. In an embodiment, cloud keys are embedded into text configuration files, structured configuration files (e.g., JSON, YAML, XML, etc.), scripts, source code, and the like. Example implementations of cloud keys include AWS IAM access keys, OAuth® refresh tokens, access tokens, and the like.

[0066] By generating a security graph 300 including such nodes and populating it with data representing the cloud computing environment allows assessing of cybersecurity risks. For example, if a first cloud key is compromised, it is readily apparent what other objects are vulnerable as a result, by querying the security graph 300 and detecting cloud entities which are represented by nodes connected to, for example, a node representing the first cloud key. In an embodiment each node further stores metadata and data relating to the object. For example, a cloud key node 320 may include therein a unique account identifier.

[0067] In some embodiments, the security graph 300 further includes a representation of a cybersecurity issue. For example, a misconfiguration is represented by a node in the security graph, in an embodiment. In an embodiment a node representing a cybersecurity issue is connected to a node which represents a resource. This is performed to indicate that the resource includes the cybersecurity issue. For example, an inspector is configured to detect a cybersecurity issue, and detects the cybersecurity issue on a software container which is inspected by the inspector. In an embodiment, a security graph is updated to include a node representing the software container connected to a node representing the cybersecurity issue.

[0068] In certain embodiments, generating a node representing a cybersecurity issue allows to reduce redundant information stored in a graph database, where storing a connection requires less resources than storing information about the cybersecurity issue in each node representing a resource where the cybersecurity issue is detected. This allows compact representation, thereby reducing computer resource consumption. This further allows to rapidly detect all resources having a certain cybersecurity issue, as rather than querying each node to determine if the node includes information on a specific

cybersecurity issue, a single node is queried to detect nodes connected to it. This reduces the amount of processing required on a database search.

[0069] Fig. 4 is a schematic illustration of a code inspector 180 implemented according to an embodiment. The code inspector 180 may be implemented as a physical machine or a virtual workload, such as a virtual machine or container.

[0070] When implemented as a physical machine, the code inspector 180 includes at least one processing circuity 410, for example, a central processing unit (CPU). In an embodiment, the processing circuity 410 may be, or be a component of, a larger processing unit implemented with one or more processors. The one or more processors may be implemented with any combination of general-purpose microprocessors, microcontrollers, digital signal processors (DSPs), field programmable gate array (FPGAs), programmable logic devices (PLDs), controllers, state machines, gated logic, discrete hardware components, dedicated hardware finite state machines, or any other suitable entities that can perform calculations or other manipulations of information. In certain embodiments it may be advantageous for the at least one processing circuity 410 to further include one or more general purpose graphic processor units (GPGPUs). For example, for comparing and generating digests, a GPGPU may have improved performance over a CPU.

[0071] The processing circuity 410 is coupled via a bus 405 to a memory 420. The memory 420 may include a memory portion 425 that contains instructions that when executed by the processing element 410 performs the method described in more detail herein. The memory 420 may be further used as a working scratch pad for the processing element 410, a temporary storage, and others, as the case may be. The memory 420 may be a volatile memory such as, but not limited to random access memory (RAM), or non-volatile memory (NVM), such as, but not limited to, Flash memory. The memory may further include a memory portion 425 which is used to store objects extracted from a configuration code.

[0072] The processing element 410 may be coupled to a network interface controller (NIC) 430, which provides connectivity to one or more cloud computing environments, such as the first cloud computing environment 140 of Fig. 1, via a network.

[0073] The processing element 410 may be further coupled with a storage 440. The storage 440 may be used for the purpose of holding a copy of the method executed in accordance with the disclosed technique. The storage 440 may include a storage portion 445 containing a configuration code for deployment in a cloud computing environment.

[0074] The processing element 410 and/or the memory 420 may also include machine-readable media for storing software. Software shall be construed broadly to mean any type of instructions, whether referred to as software, firmware, middleware, microcode, hardware description language, or otherwise. Instructions may include code (e.g., in source code format, binary code format, executable code format, or any other suitable format of code). The instructions, when executed by the one or more processors, cause the processing system to perform the various functions described in further detail herein.

[0075] Fig. 5 is an example of a code object, shown in accordance with an embodiment. A code object 500 includes an object type 510. The object type 510 indicates, in this example, that this code object is a resource type, i.e. executing instructions related to this object will deploy a resource in a cloud computing environment. The object type further includes data fields, such as instance type data field 512 and network association data field 514. The instance type 512 specifies what type of resource is to be deployed, in this case the instance type is a t2.micro, which is a processing instance used in the AWS cloud computing environment. The network association field 514 indicates, in this example, that the instance should be associated with a specific virtual private cloud (VPC). In this example the code object is a data structure having parameters (or data fields) which can be customized to generate resources, accounts, and the like, in a cloud computing environment.

[0076] The various embodiments disclosed herein can be implemented as hardware, firmware, software, or any combination thereof. Moreover, the software is preferably implemented as an application program tangibly embodied on a program storage unit or computer readable medium consisting of parts, or of certain devices and/or a combination of devices. The application program may be uploaded to, and executed by, a machine comprising any suitable architecture. Preferably, the machine is implemented on a computer platform having hardware such as one or more central processing units ("CPUs"), a memory, and input/output interfaces. The computer platform may also include

an operating system and microinstruction code. The various processes and functions described herein may be either part of the microinstruction code or part of the application program, or any combination thereof, which may be executed by a CPU, whether or not such a computer or processor is explicitly shown. In addition, various other peripheral units may be connected to the computer platform such as an additional data storage unit and a printing unit. Furthermore, a non-transitory computer readable medium is any computer readable medium except for a transitory propagating signal.

[0077] All examples and conditional language recited herein are intended for pedagogical purposes to aid the reader in understanding the principles of the disclosed embodiment and the concepts contributed by the inventor to furthering the art, and are to be construed as being without limitation to such specifically recited examples and conditions. Moreover, all statements herein reciting principles, aspects, and embodiments of the disclosed embodiments, as well as specific examples thereof, are intended to encompass both structural and functional equivalents thereof. Additionally, it is intended that such equivalents include both currently known equivalents as well as equivalents developed in the future, i.e., any elements developed that perform the same function, regardless of structure.

[0078] It should be understood that any reference to an element herein using a designation such as “first,” “second,” and so forth does not generally limit the quantity or order of those elements. Rather, these designations are generally used herein as a convenient method of distinguishing between two or more elements or instances of an element. Thus, a reference to first and second elements does not mean that only two elements may be employed there or that the first element must precede the second element in some manner. Also, unless stated otherwise, a set of elements comprises one or more elements.

[0079] As used herein, the phrase “at least one of” followed by a listing of items means that any of the listed items can be utilized individually, or any combination of two or more of the listed items can be utilized. For example, if a system is described as including “at least one of A, B, and C,” the system can include A alone; B alone; C alone; 2A; 2B; 2C; 3A; A and B in combination; B and C in combination; A and C in combination; A, B, and C in combination; 2A and C in combination; A, 3B, and 2C in combination; and the like.

CLAIMS

What is claimed is:

1. A method for detecting a vulnerable code object in configuration code for deploying instances in a cloud computing environment, comprising:

accessing a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance;

querying a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and

generating a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

2. The method of claim 1, further comprising:

generating an alert based on the cybersecurity issue.

3. The method of claim 1, further comprising:

deploying a virtual instance based on the first code object in response to determining that the detected node is not connected to a node representing a cybersecurity issue.

4. The method of claim 1, further comprising:

detecting in the configuration code a data field of a code object having any one of: a string, and a value.

5. The method of claim 4, wherein the data field represents any one of: a private key identifier, a public key identifier, a policy identifier, a user account identifier, a service account identifier, a version identifier, and a resource type identifier.

6. The method of claim 1, further comprising:

generating a query directed at the security graph, the query including an identifier of the first code object.

7. The method of claim 6, wherein the identifier is any one of: a user account identifier, a service account identifier, a version identifier, and a resource type identifier.

8. The method of claim 1, further comprising:

traversing the security graph to detect a node, wherein the node represents the first code object, wherein a node representing the first code object includes a data field value which matches with a data field value of the first code object.

9. The method of claim 1, wherein the cybersecurity risk is represented by a risk node in the security graph, and the detected node is connected to the risk node.

10. A non-transitory computer readable medium having stored thereon instructions for causing a processing circuitry to execute a process, the process comprising:

accessing a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance;

querying a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and

generating a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

11. A system for detecting a vulnerable code object in configuration code for deploying instances in a cloud computing environment, comprising:

a processing circuitry; and

a memory, the memory containing instructions that, when executed by the processing circuitry, configure the system to:

access a configuration code, the configuration code including a plurality of code objects, each code object of the plurality of code objects corresponding to a deployable virtual instance;

query a security graph to detect a node having an attribute value which matches a value extracted from a first code object of the plurality of code objects, wherein the security graph includes a representation of a cloud computing environment; and

generate a mitigation action in response to determining that the detected node is associated with a cybersecurity issue.

12. The system of claim 11, wherein the memory includes further instructions which when executed by the processing circuitry further configure the system to:

generate an alert based on the cybersecurity issue.

13. The system of claim 11, wherein the memory includes further instructions which when executed by the processing circuitry further configure the system to:

deploy a virtual instance based on the first code object in response to determining that the detected node is not connected to a node representing a cybersecurity issue.

14. The system of claim 11, wherein the memory includes further instructions which when executed by the processing circuitry further configure the system to:

detect in the configuration code a data field of a code object having any one of: a string, and a value.

15. The system of claim 14, wherein the data field represents any one of: a private key identifier, a public key identifier, a policy identifier, a user account identifier, a service account identifier, a version identifier, and a resource type identifier.

16. The system of claim 11, wherein the memory includes further instructions which when executed by the processing circuitry further configure the system to:

generate a query directed at the security graph, the query including an identifier of the first code object.

17. The system of claim 16, wherein the identifier is any one of: a user account identifier, a service account identifier, a version identifier, and a resource type identifier.

18. The system of claim 11, wherein the memory includes further instructions which when executed by the processing circuitry further configure the system to:

traverse the security graph to detect a node, wherein the node represents the first code object, wherein a node representing the first code object includes a data field value which matches with a data field value of the first code object.

19. The system of claim 11, wherein the cybersecurity risk is represented by a risk node in the security graph, and the detected node is connected to the risk node.

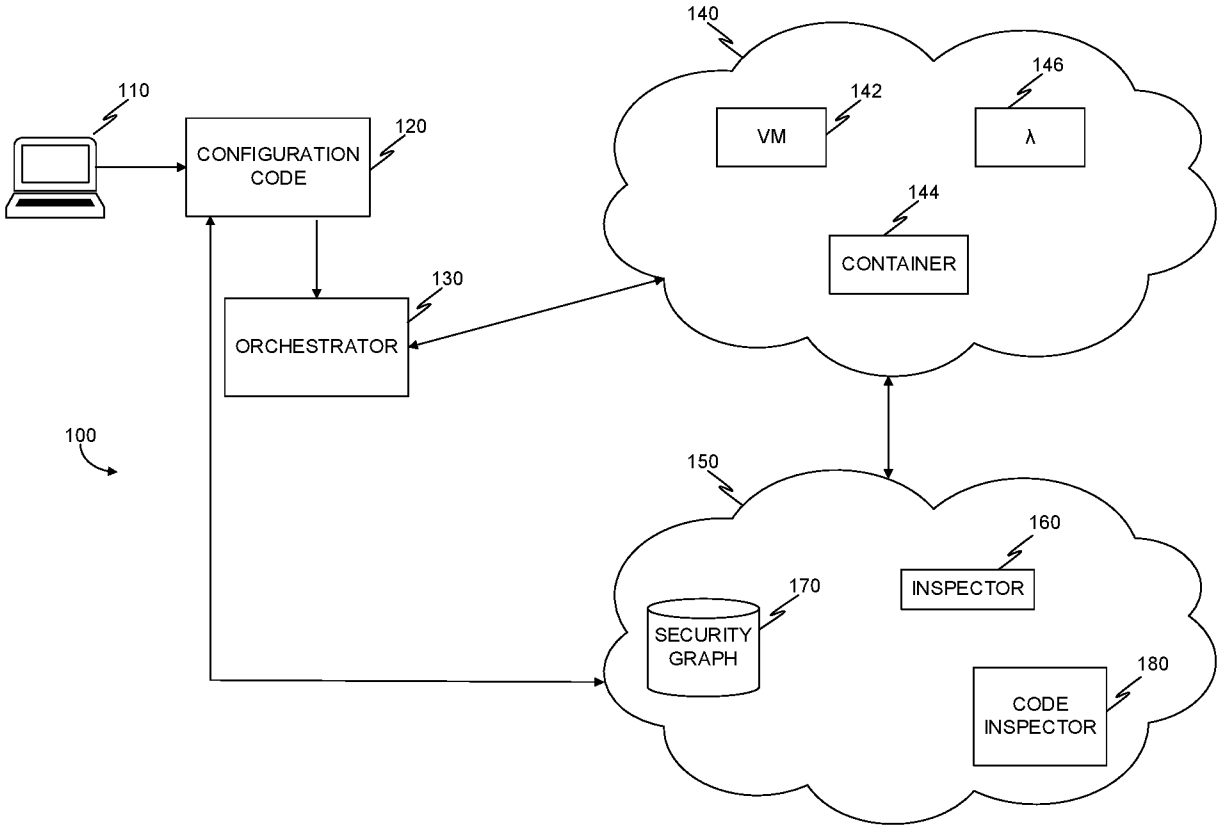

140
146
110
142
120
VM
λ
CONFIGURATION CODE
144
CONTAINER
130
ORCHESTRATOR
100
150
160
170
INSPECTOR
SECURITY GRAPH
180
CODE INSPECTOR

FIGURE 1

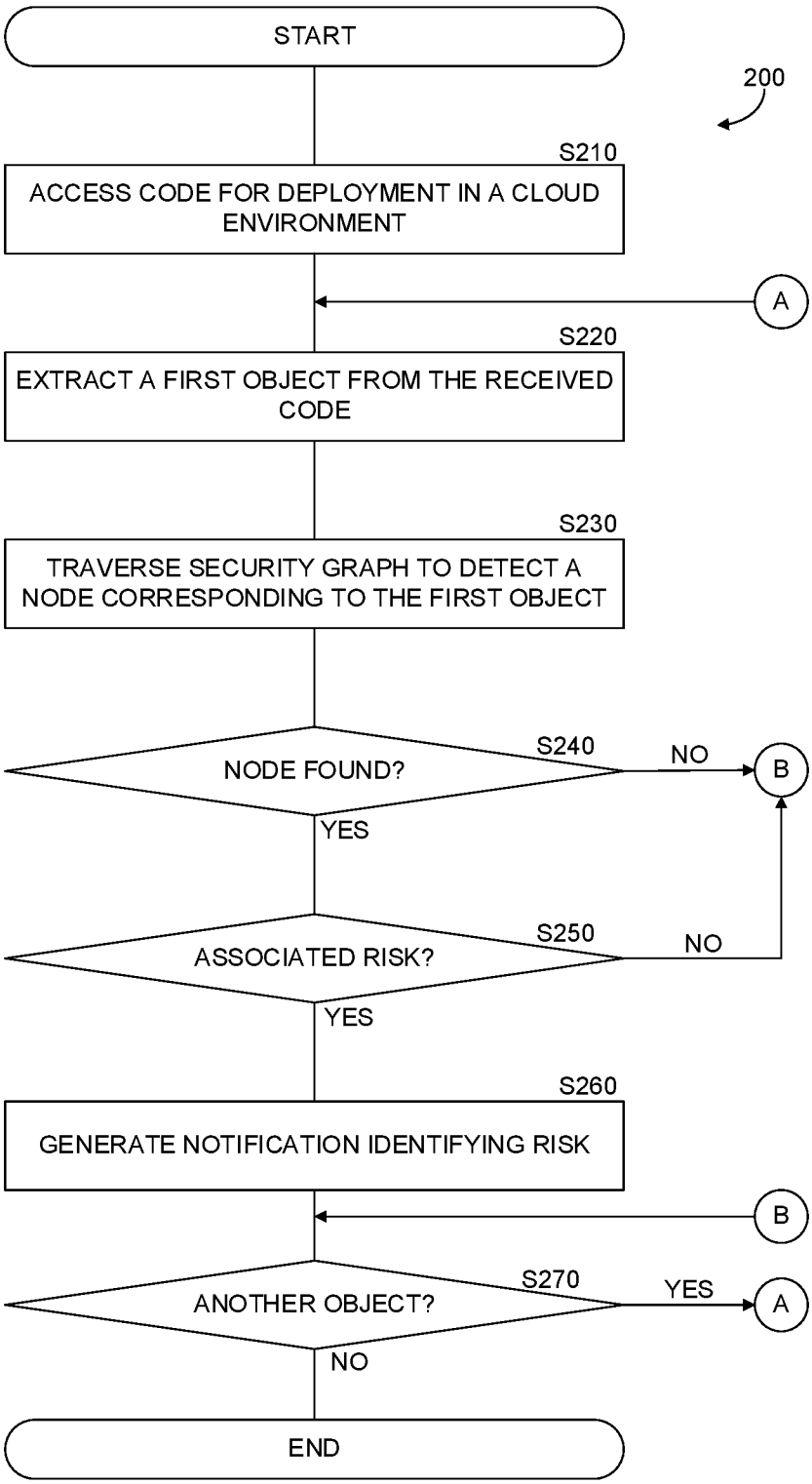

START
200
S210
ACCESS CODE FOR DEPLOYMENT IN A CLOUD ENVIRONMENT
A
S220
EXTRACT A FIRST OBJECT FROM THE RECEIVED CODE
S230
TRAVERSE SECURITY GRAPH TO DETECT A NODE CORRESPONDING TO THE FIRST OBJECT
S240
NODE FOUND?
NO
B
YES
S250
ASSOCIATED RISK?
NO
YES
S260
GENERATE NOTIFICATION IDENTIFYING RISK
B
S270
ANOTHER OBJECT?
YES
A
NO
END

FIGURE 2

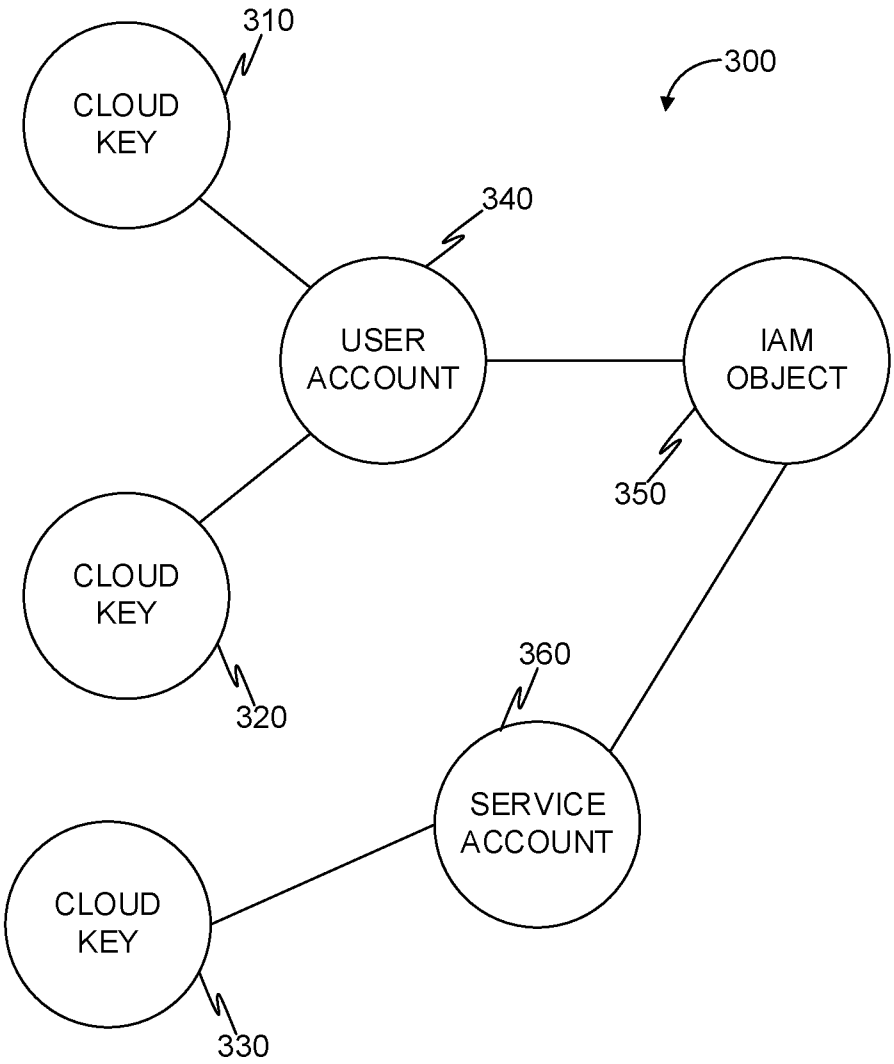

310
300
CLOUD KEY
340
USER ACCOUNT
IAM OBJECT
350
CLOUD KEY
320
360
SERVICE ACCOUNT
CLOUD KEY
330

FIGURE 3

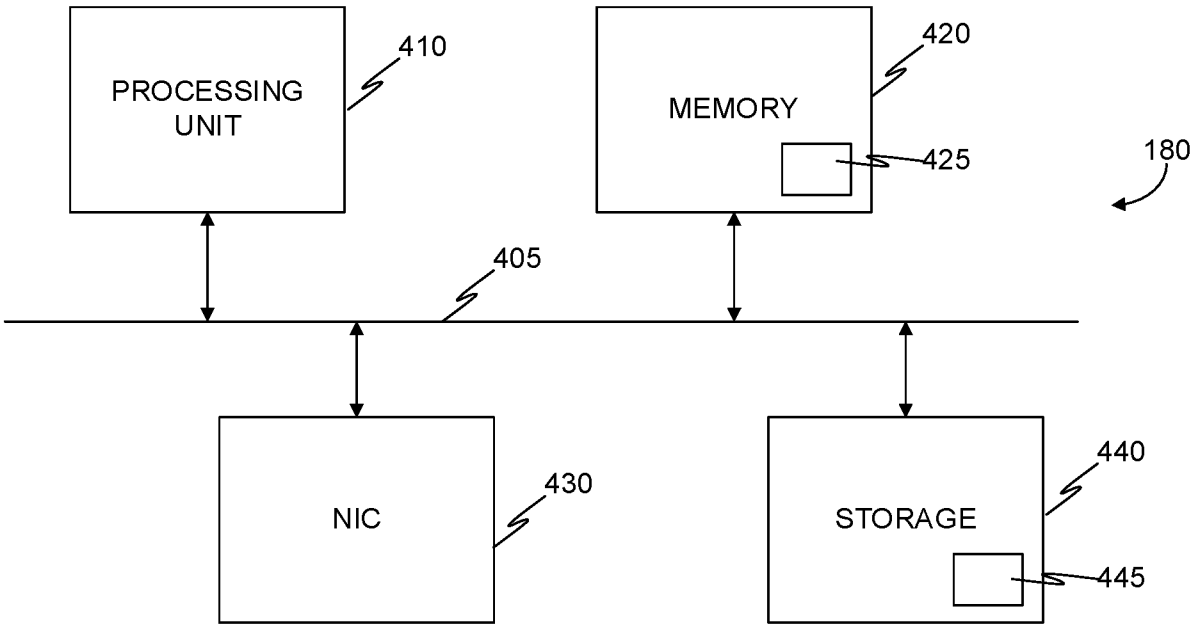

PROCESSING UNIT
410
MEMORY
420
425
180
405
NIC
430
STORAGE
440
445

FIGURE 4

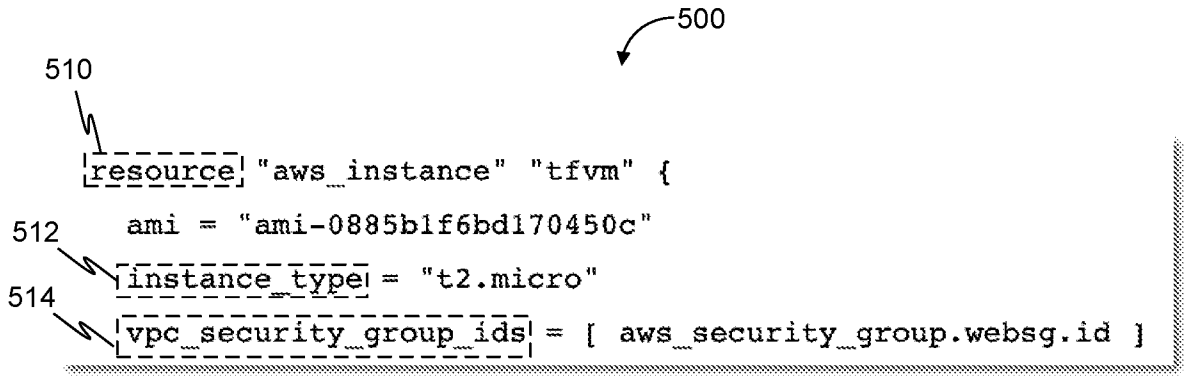

500
510
resource "aws_instance" "tfvm" {
ami = "ami-0885b1f6bd170450c"
512
instance_type = "t2.micro"
514
vpc_security_group_ids = [aws_security_group.websg.id]

FIGURE 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2022/060940

A. CLASSIFICATION OF SUBJECT MATTER

G06F 21/57(2023.01)i; ***H04L 9/40***(2023.01)i
CPC:G06F 21/577; H04L 63/20; H04L 63/1433

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 21/57; H04L 9/40
CPC:G06F 21/577; H04L 63/20; H04L 63/1433

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Databases consulted: Esp@cenet, Google Patents, Google Scholar, FamPat database, Orbit, Similari (AI-based) Search terms used: Cloud Technologies, AWS Cloud Management, Multi-Cloud, Cloud Security, infrastructure, code, graph, vulnerabilities, deploy, workload, configuration, environment, inspect, detect, Iac.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2017026416 A1 (RAYTHEON CO) 26 January 2017 (2017-01-26) Entire document	1-8,10-18
A	US 2020244678 A1 (ORCA SECURITY LTD) 28 July 2020 (2020-07-28) Entire document	1-8,10-18
A	US 2008221833 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 11 September 2008 (2008-09-11) Entire document	1-8,10-18

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 January 2023	**01 February 2023**

Name and mailing address of the ISA/IL	Authorized officer
Israel Patent Office **Technology Park, Bldg.5, Malcha, Jerusalem, 9695101, Israel** **Israel** Telephone No. **972-73-3927164** Email: **pctoffice@justice.gov.il**	**DRORI Avishag** Telephone No.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2022/060940

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☑ Claims Nos.: **9,19**
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

[0001] No search opinion could be made in respect to claims 9 and 19 for the following reasons: The term "risk node" is obscured in its scope and lack any concrete definition or exemplifications in the supporting description which would elucidate its meaning. Due to said lack of clarity, no meaningful search opinion could be established.

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/IB2022/060940

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
US 2017026416 A1	26 January 2017	US 2017026416 A1	26 January 2017
		US 10255370 B2	09 April 2019
US 2020244678 A1	28 July 2020	US 2020244678 A1	30 July 2020
		US 11431735 B2	30 August 2022
		US 2021336976 A1	28 October 2021
		US 11516231 B2	29 November 2022
		US 2020244692 A1	30 July 2020
		US 2021329019 A1	21 October 2021
		US 2021377287 A1	02 December 2021
		US 2022417270 A1	29 December 2022
US 2008221833 A1	11 September 2008	US 2008221833 A1	11 September 2008
		US 7467068 B2	16 December 2008