United States Patent Application Publication	20250258916
Kind Code	A1
Publication Date	August 14, 2025
Inventor(s)	Rowland; Craig et al.

DRIFT DETECTION IN REMOTE COMPUTER SYSTEMS

Abstract

Investigation systems and methods can investigate a target host computer. A reference computer investigation of the reference host computer can be performed by the investigation system sending at least one investigative module to a reference host computer, and the at least one investigative module performing the at least one investigative function and returning reference investigation data. A target computer investigation of the target host computer can be performed by the investigation system sending the at least one investigative module to the target host computer, and the at least one investigative module performing the at least one investigative function and returning comparison investigation data. A reference data fingerprint of the reference investigation data is compared to a comparison data fingerprint of the comparison investigation data to determine if the reference and comparison data fingerprints are identical.

Inventors: **Rowland; Craig (Christchurch, NZ), Wilson; Matthew (Hillsboro, OR)**

Applicant: **Sandfly Security Limited** (Christchurch, NZ)

Family ID: **1000008491871**

Appl. No.: **19/053077**

Filed: **February 13, 2025**

Foreign Application Priority Data

NZ	808200	Feb. 13, 2024

Publication Classification

Int. Cl.: **G06F21/56** (20130101); **H04L9/40** (20220101)

U.S. Cl.:

CPC **G06F21/566** (20130101); **H04L63/1416** (20130101); **H04L63/1433** (20130101);

Background/Summary

CROSS REFERENCE TO RELATED APPLICATION

[0001] The present application claims the benefit of New Zealand provisional patent application Ser. No. 808200, filed Feb. 13, 2024, which is hereby incorporated herein by reference in its entirety.

FIELD OF THE DISCLOSURE

[0002] The disclosure relates to methods, systems and software for investigating computer systems for intrusion, malware or other threats.

[0003] In particular, the disclosure relates to an investigation system, method and software for investigating remote computers by performing agentless drift detection.

BACKGROUND

[0004] Computer security software is used herein as a generalised term to describe computer programs with anti-virus, anti-malware, anti-phishing, anti-intrusion or other functionality used to maintain computer operability and data-security. Various forms of computer security software are well-known and typically run on a host computer to scan for intrusion, malware or other problems on the host computer.

[0005] Often such security software has a high-performance load on the system as the software requires use of the central processing unit (CPU) and random-access memory (RAM) to scan many files, services and processes to ensure that the computer is not compromised. Some software may be run on user demand, on a schedule or upon certain parameters being met, e.g. the CPU load on the computer is low.

[0006] In computer servers, both security and performance are a high priority and security scans are typically scheduled to run when the server is not under high load or at times when the server is not expected to be under high demand.

[0007] Most security software is run with a computer program installed on the host computer to be scanned, (i.e. this computer program is known in the art as a “software agent”). The software agent is either fully capable of conducting security scans without other data or conducting scans using data received from another computer, e.g. from a master server.

[0008] Software agents are notoriously unreliable and can cause problems for network administrators e.g. software agents require constant updates and can cause conflicts with other system updates that need to be installed. Agents, even when idle, also constantly use a proportion of the capacity of the CPU, RAM, and other system resources. A software agent that requires an average of a relatively conservative 5% CPU time means the operator of the host computer may need to operate 5% more host computers (e.g. servers) to perform the tasks required due to the occupied CPU resource. This is a significant cost for a large enterprise.

[0009] Furthermore, when agents need to be updated, administrators must update the agent on every single host computer that agent is installed on. This process is fraught with problems and requires extensive testing to be sure the updates will not cause conflicts or failures with other loaded software. Moreover, there is considerable risk in automated deployment and upgrade of agents across the network. Any problem caused by the agent or upgrade may impact countless hosts and cause serious disruptions to network operations.

[0010] Investigation systems that require an agent on the host computer also have the disadvantage of potentially alerting intruders to the agent's presence and therefore warning the intruder that security is present. The intruder may then take steps to avoid detection, disable the agent or provide false data to the agent and associated software.

[0011] The presence of an agent on the host computer provides information to an intruder about the security of the system and provides an avenue for the intruder to circumvent that security by

disabling the agent or local software or analysing it to determine when security scans are scheduled and running outside the schedule times while hiding or removing themselves during scheduled scans.

[0012] Most pre-existing local systems aim to provide (with respect to the time domain), a contiguous threat defence scheduled at regular intervals, which, while potentially comprehensive, applies a significant performance burden on the hardware and provides a clear and easily detectable target and time for an intruder to circumvent.

[0013] Similarly, prior art agentless security systems either run on a schedule or on demand and use contiguous scanning and thus become predictable for the intruder and enable the intruder to 'hide' any evidence while the system is scanning and then become active when the security system is dormant or otherwise inactive.

[0014] Contiguous scanning such as in the prior art is convenient for the operator as they can simply schedule a scan, normally at a time when system load is presumed to be low. However, as described previously, contiguous scanning presents a relatively high-performance load for the host computer and provides increased likelihood that an intruder may avoid detection by removing evidence prior to a scan.

[0015] In contrast, "agentless" approaches to security require no software to be loaded on the host that needs protection, and thus have no resource impact on the host computer when not active.

[0016] A conventional agentless approach involves obtaining remote access to the host computer, mounting the file system, and running scans of that file system using the computer that remotely accessed the host.

[0017] However, remote access file scans have disadvantages including: [0018] Bandwidth usage—the scanning device must maintain the connection while scanning, leading to network bandwidth consumption. [0019] Resource load/limit—the scanning device uses an amount of CPU, RAM, network and disk resource during the course of the scan. This means that the number of clients that can be scanned at any one time is limit by the resources available to the scanning device. [0020] Higher chance of being detected—any intruder active on the client device may detect the scan connection and take action to avoid detection, e.g. delete malicious files from client.

[0021] A solution was presented to these problems in commonly-owned U.S. Pat. No. 12,058,149, entitled COMPUTER INVESTIGATION METHOD AND SYSTEM, issued on Aug. 6, 2024, which is incorporated by reference herein in its entirety.

[0022] Malware is constantly changing, along with the systems targeted. It is often difficult to identify zero-day malware or other malware that is not known or is not easily identifiable. Various techniques are utilised in the prior art for addressing this issue, including: [0023] a) Using existing known malware behaviour as a reference and searching for similar behaviour. [0024] b) Searching for system interactions that are characteristic of malware or result in a compromised system. [0025] c) Using artificial intelligence (AI) and Machine Learning to determine what is considered a safe operating condition of the system.

[0026] However, such techniques are not perfect and can struggle to detect some forms of malware that appear innocuous.

[0027] Many computer systems have a predictable and stable configuration. Such systems for example include embedded devices, appliances, Internet of Things devices, or servers running a fixed set of services.

[0028] Changes to the files in such stable systems can indicate a malware compromise and some prior art malware detection methods may look for changes to files that indicate malware is present. These methods are known as 'system integrity monitoring', 'file integrity monitoring', 'file change alerting' or similar. However, such methods are limited to understanding little more than what the file is and that the file has changed. Without understanding the content or context of the file, innocuous changes may trigger an alert or quarantine. Moreover, there may be no way to avoid false positives on the file change without excluding the entire file from the scan, which results in

more meaningful changes going undetected.
[0029] Existing file integrity monitoring systems also require a remote access connection or a software agent to run on the computer system to monitor for file changes.
SUMMARY
[0030] Investigation methods and systems can be used for investigating computer systems are used for detecting potential vulnerabilities that don't rely on reference to known malware signatures. Further, investigation methods and systems can be used for system integrity monitoring that reduce false positive results and provide additional context to detected changes. In some examples, investigation methods and systems can be used for system integrity monitoring that don't require a software agent, i.e. an agentless monitoring system.
[0031] Reference throughout the specification is made to the present disclosure as relating to software, methods and systems for investigating unauthorised intrusion or malware on remote servers, although this should not be seen as limiting and the principles of the present disclosure may be applied to any computer system or computing device, including mobile devices and may be utilised for other computing applications.
[0032] To aid brevity and clarity, reference herein will be made to hardware devices in the singular, however, such reference should be interpreted to also include multiple components forming the device and/or multiple devices sharing the function, e.g. reference herein to a “server” should be interpreted to include multiple servers, distributed servers, cloud-based servers and the like.
[0033] As used herein the term “software” refers to one or more computer programs.
[0034] As used herein the term “program” or “computer program” refers to computer readable instructions (“code”) operable to provide input to a computer processor to provide output to instruct the computer to perform one or more functions. The computer program may include code in one or more programming languages or may include machine code and the form of code should not be considered to be limited.
[0035] Reference throughout this specification to the singular should be interpreted to include the plural and vice versa unless specifically stated otherwise.
[0036] Hereon, the terms “scan” and “investigate” with reference to a host computer may be used interchangeably to refer to the process of investigating a host computer.
[0037] As used herein the term “user account” refers to data in the form of a username or id that is deemed by the host computer to have access to resources on the host computer. The user account may relate to a human or virtual user and may have defined privileges. An individual human or virtual user may have multiple user accounts. A “user account” typically includes at least a user id and may also include other data such as name, email or other identifying information.
[0038] As used herein, the term “data form” refers to any data or process of the host computer. A data form may include any data or software, process, manipulation, transmission, operation, deletion or part thereof associated with the host computer, including active, dynamic or static data or processes. Examples of “data forms”, may include files, processes, user accounts, system hardware information, system operating system information and other system data.
[0039] As used herein, the term “data form attribute” refers to any attribute, property, value or descriptor of a data form, e.g. a file data form may have data form attributes including filename, content, timestamp, user id, group id, size, mode and path.
[0040] As used herein, the term “remote” with respect to a given host computer should not be interpreted to mean geographically remote but rather where the given host computer receives data and/or instructions by components of the investigation system that operate on a different server, computer or CPU to the given host computer or with distributed terminals, processors, software and memory resources.
[0041] As used herein, the term “intruder” refers to an unauthorised entity, system or device that attempts to, or succeeds in, accessing resources of a host computer, the resources including data, programs, applications, processor calculations, memory or any other resource of the host computer.

[0042] The term “software agent”, as used herein, refers to a computer program that resides on a host computer and is executable to operate specifically for another given computer program such that the software agent is required to be present for the given computer program to instruct the host computer to perform operations and the software agent is unable to perform those operations without the computer program. In contrast, any computer programs that are capable of being executed by the host computer to perform other operations (that are not specifically and directly related to the given computer program or data) are not considered software agents.
[0043] Thus, the term “agentless”, with respect to a computer program, refers to the nature of a computer program that may run on a host computer without requiring a software agent on that host computer.
[0044] As noted above, it should be appreciated that a given computer program may still be considered “agentless” despite utilising computer software on the host-computer if that host computer software is not provided specifically to directly operate the given computer program. In contrast to methods that require permanent and constantly running software agents on the remote host computer, such as Cylance™ (https://www.cylance.com), CarbonBlack™ (https://www.carbonblack.com/), CrowdStrike™ (https://www.crowdstrike.com/), Tanium™ (https://www.tanium.com/).
[0045] As used herein, the terms “execute” and “run”, “executing” and “running”, “execution” and “running” are used interchangeably and should be interpreted to be equivalents.
[0046] As used herein, the term “data network” should be understood to refer to any electronic network having a control system capable of receiving and transmitting data from connected computer terminals. A data network may thus be considered to include virtual private networks (VPN), the internet, telephone/cellular networks, local area networks (LAN), wide area networks (WAN), wireless networks, Wi-Fi, satellite, radio, ultra high frequency (UHF), very high frequency (VHF), infrared, mesh networks, Bluetooth, ZigBee or any other network having one or more control systems and connected terminals.
[0047] As used herein, the term “non-contiguous” with respect to the running of multiple investigative modules should be understood to refer, with respect to the time domain, to running the investigative modules independently to each other. In contrast, a “contiguous” running of multiple investigative modules involves running all investigative modules together, or sequentially without significant or discernible delay.
[0048] As used herein, the term “compare” means to perform a computer data comparison and includes any known methods for evaluating two sets of data to determine if the two sets are the same or exceed a threshold level of similarity. Computer data comparisons include string, number, date, hash, logic matches and other comparisons.
[0049] According to a first aspect of the present disclosure there is provided a method of investigating a target host computer, the method using an investigation system including a computer system with a computer processor coupled to a system memory and programmed with computer readable instructions, the investigation system being remote to the target host computer, the method using at least one investigative module, [0050] wherein the at least one investigative module includes an agentless computer program including computer readable instructions to:
[0051] perform at least one investigative function on a host computer, the at least one investigative function relating to at least one data form, [0052] return investigation data relating to the at least one investigative function, the investigation data including at least one data form attribute relating to the at least one data form, and [0053] wherein the method includes: [0054] a) performing a reference computer investigation of the reference host computer, the computer investigation including: [0055] the investigation system establishing a connection between the investigation system and the reference host computer, and [0056] the investigation system sending the at least one investigative module to the reference host computer, the at least one investigative module performing the at least one investigative function and returning reference investigation data; [0057]

b) performing a target computer investigation of the target host computer, the target computer investigation including: [0058] the investigation system establishing a connection between the investigation system and the target host computer, and [0059] the investigation system sending the at least one investigative module to the target host computer, the at least one investigative module performing the at least one investigative function and returning comparison investigation data; [0060] c) comparing a reference data fingerprint to a comparison data fingerprint to determine if the reference and comparison data fingerprints are identical, and [0061] wherein the reference data fingerprint includes at least two data form attributes of the reference investigation data, and [0062] wherein the comparison data fingerprint includes at least two corresponding data form attributes of the comparison investigation data.

[0063] The target host computer may be the same as the reference host computer, i.e. to detect drift detection over time for the target computer. Alternatively, the target host computer may be different to the reference host computer, i.e. the reference host computer may be a computer in a state that is known to be malware-free and thus the target host computer can be compared to the reference host computer to detect drift from the expected state.

[0064] For example, the reference data fingerprint includes a hash of the at least two data form attributes of the reference investigation data, and the comparison data fingerprint includes a hash of the at least two corresponding data form attributes of the comparison investigation data.

[0065] For example, the data fingerprint is included of a combination or concatenation of at least two data form attributes.

[0066] For example, the reference data fingerprint includes a hash of the combination or concatenation of the at least two data form attributes of the reference investigation data, and the comparison data fingerprint includes a hash of the combination or concatenation of the at least two corresponding data form attributes of the comparison investigation data.

[0067] While the data fingerprints may contain other data, in one preferred embodiment, at least one the data fingerprint is included of only: [0068] the corresponding at least two data form attributes, or [0069] a combination or concatenation of the corresponding at least two data form attributes, or [0070] a hash of the corresponding at least two data form attributes, or [0071] a hash of the combination or concatenation of the corresponding at least two data form attributes.

[0072] The investigation system can perform system integrity monitoring, or ‘drift-detection’ by using time-delimited investigations and comparisons of the returned reference and comparison data fingerprints to determine if there are any differences therebetween. The differences can indicate the potential presence of malware, a compromised target host computer or some other potential problem on the host computer.

[0073] The reference data fingerprint and comparison data fingerprint use the same set of data form attributes, e.g. the data fingerprints may be formed from the combination of the values of the filename and path attributes in each corresponding investigation.

[0074] For example, multiple reference investigations can be performed to create multiple corresponding reference data fingerprints, each reference data fingerprint representing a different set or collation of data form attributes. The multiple reference data fingerprints can thereby collectively form a system reference data fingerprint that represents many different data form attributes of the reference host computer.

[0075] For example, the hash is a cryptographic hash. For example, the hash function used to generate a hash from the multiple data attributes may use a SHA-512 algorithm or other hashing algorithm.

[0076] For example, the reference and comparison investigations are performed agentlessly, i.e. not requiring a software agent to be present on the corresponding host computer when the investigation system establishes the connection.

[0077] For example, the first investigative module is a binary program. The binary program is configured to run on the host computer to perform investigative functions, including the first

investigative function.
[0078] For example, in the computer investigations, the at least one investigative modules include a first investigate module and a second investigative module, the second investigative module including data readable by the first investigative module, the data including instructions or data necessary for the first investigative module to perform the at least one investigative function.
[0079] For example, the second investigative module includes data including an identifier of an investigative function to run, and optionally a parameter(s) for the investigative function.
[0080] The second investigative module may include one or more functions, executable by the first investigative module.
[0081] The second investigative module may be sent sequentially or simultaneously with the first investigative module. The second investigative module may be sent before or after the first investigative module.
[0082] The data fingerprint may include a hash of a subset of the investigation data or may be a hash of the entirety of the investigation data, i.e. the data fingerprint may include some, or all, of the corresponding data form attributes.
[0083] For example, the data fingerprints include hashes representing the corresponding investigation data or subset thereof.
[0084] For example, the at least one investigative module performs hashing of the at least two data form attributes to produce the data fingerprint. The data fingerprints can thus be sent to the investigation system without sending the raw investigation data-which may be large or contain sensitive information.
[0085] It will be appreciated that the investigation data may already contain hashes of files, processes or other data and thus the data fingerprint may include a hash of other hashed data.
[0086] For example, the investigation system stores data records, the data records including a record containing: [0087] the data fingerprint, [0088] an identifier of the corresponding host computer, and [0089] an identifier of the investigative module run to produce the data fingerprint.
[0090] For example, the investigation system sends multiple different investigative modules to the corresponding host computers, thereby performing multiple different investigation functions, each the investigative module returning a corresponding data fingerprint that is collected by the investigation system.
[0091] In an alternative embodiment, the investigation system may produce a single data fingerprint representing the investigation data returned by multiple different investigative modules. The data fingerprint may be a representation of the combination of all, or part, of the investigation data returned by the multiple different investigative modules.
[0092] For example, the investigation system stores the data records in a database. The database may be stored in a computer-readable medium on the investigation system or may be stored remotely on a separate system or data store.
[0093] In some applications users may want to be able to compare changes in different subsets of the investigation data rather than the entirety of the data. Thus, for example, the investigative module is configured to generate multiple reference data fingerprints, each reference data fingerprint corresponding to a different subset of the investigation data.
[0094] For example, the investigation system stores the multiple reference data fingerprints for each subset of investigation data, including at least: [0095] a reference data fingerprint of a first subset of the first investigation data, and [0096] a reference data fingerprint of a second subset the first investigation data,
wherein the first and second subsets are different.
[0097] In one embodiment, the second subset may be a superset of the first subset.
[0098] For example, the subsets of first investigation data correspond to different data forms and/or different sets of data form attributes.
[0099] The investigation module may perform multiple comparisons, each comparison comparing

data fingerprint pairs for a different subset of the investigation data. Users may not want to be alerted to all changes in the target host computer. Some subsets of data for example, may be expected to show changes over time or a user may not want to be alerted to changes in that subset of data. Different data fingerprint comparisons may thereby be chosen to provide varying levels of monitoring according to the user's needs.

[0100] The investigation system may include multiple computers, the connection taking place between the corresponding host computers and any one of the investigation system computers. Processing and data storage may take place on the same or other of the investigation system computers.

[0101] The reference host computer may be local to the investigation system in some cases and/or the data fingerprint of the local host computer may be stored in the investigation system.

[0102] The investigative modules may be configured to perform various investigative functions depending on the application and requirements of the investigation. Users of the investigation system may want to perform different levels of investigation and may only require detection of changes in data fingerprints corresponding to specific investigative modules, data forms and/or data form attributes. The combination of different investigative modules, data forms and/or data form attributes is herein referred to as a ‘reference data fingerprint’,

[0103] For example, a user may specify different change monitoring ‘profiles’. For example, a user may specify a permissive and moderate profiles, with the permissive profile containing a subset of the data form attributes of the moderate profile.

[0104] Examples of the investigative modules and corresponding investigative functions that mya be used with the present disclosure are now described.

[0105] An investigative module may include an ‘SSH Hunter’ investigative module, configured to locate and collect investigation data corresponding to a data form including a secure shell (SSH) Public key to ascertain if there are any user accounts of the host computer that have data forms, the data forms including at least one authentication token in the form of an SSH public key.

[0106] For example, the SSH hunter investigative module is configured to run on the host computer to locate the SSH public key by performing the following steps: [0107] determining at least one user account of the host computer, and [0108] locating at least one authentication key file such as, for example, associated with the user account.

[0109] For example, the SSH hunter investigative module is configured to run on the host computer to collect investigation data relating to the SSH public key by performing the following steps: [0110] reading the authentication key file and determine if any data in the authentication key file is, or relates to, an SSH key, and [0111] returning investigation data relating to the SSH key.

[0112] The ‘SSH hunter’ investigative module may thus act to locate and retrieve information about SSH keys on host computers. This information can be used to identify potential threats or vulnerabilities, for example, an SSH public key may be present that provides a malicious user access to the host computer.

[0113] As the investigation system is agentless, the ‘SSH hunter’ investigative module can be run on myriad different devices, even if those devices don't have software agents present. This is highly beneficial for devices where deploying a software agent is unfeasible or undesirable, e.g., in network infrastructure with many servers or low-powered host computers such as IoT and embedded devices.

[0114] The ‘SSH hunter’ investigative module thus presents a significant security advantage as it enables a user of the investigation system to easily locate and retrieve SSH keys, across many devices and many types of devices, with minimal resource usage and all with no agent deployment required.

[0115] The ‘SSH hunter’ investigative module can be run as a singular module sent to the host computer, thereby requiring only minimal processor and memory usage. In contrast, running an equivalent process using a software agent on the host requires not only the process to run but also

the software agent process to run. The ‘SSH hunter’ investigative module is thus ideal for performing SSH key investigations of IoT and other low-powered devices.
[0116] An “authentication key file” refers to any file containing authentication keys or having a filename equal to an authentication key. User accounts typically have one or more such associated authentication key files that store authentication keys for that user. However, the ‘SSH hunter’ investigative module may also be configured to search other files for SSH keys as some host computers may not use the same naming scheme for their authorized key files. Moreover, searching other files on the host computer may locate SSH keys stored for potentially malicious purposes.
[0117] For example, the investigation data includes at least the SSH Public Key itself and may also include the line number and/or other location identifier of the SSH Public Key in the file.
[0118] Authentication key files may contain hundreds or thousands of authentication keys. It is common for a user to remove an authentication key, thinking that they have revoked access, only for that key to also be present at another location in the authentication key file. This duplication can present an access risk. Thus, retrieving investigation data that includes not only whether a key is present but also where an SSH Public Key occurs in a file can be used to identify duplicates and mitigate risk.
[0119] For example, the investigation data includes data corresponding to the user account and may include user data such as name, Internet Protocol (IP) address and/or identification (ID).
[0120] For example, the investigation data includes data corresponding to the host computer and may include a host computer identifier such as name, IP address and/or other id. Alternatively, or in addition, the investigation system may record an identifier of the host computer so that the investigation data can be related to the host computer.
[0121] Thus, the investigation system can determine the user accounts and hosts associated with SSH Public Keys retrieved in the investigation data. This information can be used to identify potential threats or vulnerabilities, e.g., if an SSH key is present for multiple user accounts or how widespread a SSH key providing access to a malicious entity is.
[0122] For example, the investigation system is configured to store the investigation data in the database.
[0123] For example, the investigation system is configured to store temporal information with the investigation data. The temporal information may include timestamps of when the investigation data includes: [0124] the first occurrence of an SSH key; [0125] the first occurrence of an SSH key for a given host computer and/or user account; [0126] the first occurrence of an SSH key for a given authentication key file and/or location in the authentication key file; [0127] subsequent occurrences of an SSH key; [0128] subsequent occurrences of an SSH key for a given host computer and/or user account; [0129] subsequent occurrences of an SSH key for a given authentication key file and/or location in the authentication key file; [0130] any combination, iteration or permutation of the above.
[0131] In one embodiment, the investigation data is stored in the database in records in at least a first table and a second table, the first table including records containing the SSH keys and the second table containing records with a relation to the SSH keys, as well data corresponding to the host computer and/or user. The host and/or user data may include respective host or user data or may include an id in a relation field for relating the investigation data record to a record in another table containing host computer data and/or user account data.
[0132] For example, the investigation system includes, or has access to, one or more databases for storing data such as, data form attributes of the host computer, users and scanning nodes, investigative modules, investigation data, results from the investigative modules and other data. For example, the investigation system includes a query system for querying the investigation data, wherein the query system is configured to return query results including at least one of: [0133] an SSH key that matches an SSH key stored in a table or database containing SSH keys that match a predetermined designation, e.g. the SSH keys may be designated as ‘banned’, ‘malicious’ or other

designation indicating they should not be present on the host computer; [0134] duplicate SSH keys, i.e., SSH keys that occur more than once in a table in the database, or are related to multiple user accounts, host computers and/or authentication key files; [0135] SSH keys with timestamps within a given timeframe; [0136] SSH keys that are present in the investigation data and recorded in the database as removed or were previously marked as removed; [0137] all occurrences of an SSH key; [0138] all occurrences of an SSH key related to a user account, user account type, host computer, or host computer type; [0139] all SSH keys related to a user account, user account type, host computer, or host computer type; [0140] user accounts that have a given ‘threshold’ number of related SSH keys; [0141] user accounts related to SSH Keys that are related to a given or predefined host computer; [0142] host computers related to SSH Keys that are related to a given or predefined user account; [0143] any combination, iteration, or permutation of the above.

[0144] For example, after retrieving the investigation data from the host computer, the investigation system is configured to update records in the database with a tag indicating an SSH key has been removed if the SSH key pre-existed in the database but was not found in the investigation data. This indicates that the SSH key has been removed from the host computer.

[0145] For example, a timestamp is added to a record, indicating when the SSH key was removed.

[0146] In a further embodiment, if an SSH key tagged as ‘removed’ is present in further investigation data from subsequent investigations, a new record is created in the database for the SSH key. Thus, two records will exist for the same SSH key, the ‘removed’ record and a new ‘active’ record. Keeping records of when keys are removed and reappear is useful in providing a timeline of when the key permitted access to the corresponding user on the corresponding host.

[0147] For example, the investigation system reads the investigation data and for each SSH key included in the investigation data performs a query of the database to determine if the SSH key is already present in the database. In a further embodiment, the query, or additional query, determines if the SSH key is already present in the database and related to a host computer and/or user account.

[0148] For example, the at least one investigative module includes a computer program configured to run on the host computer using software that is non-specific to the investigative module.

[0149] In another embodiment, the at least one investigative module includes data utilizable by a computer program configured to run on the host computer using software that is non-specific to the investigative module.

[0150] It should be appreciated that the investigation system may have typical computing resources for deploying and executing software such as system memory, processors, storage memory (disk), network infrastructure and other hardware and software. It should be appreciated that these computing resources need not be in the same location, e.g. the investigation system may be a virtual system using ‘cloud-computing’ resources with distributed terminals, processors, software and memory resources.

[0151] Similarly, the database may include multiple separate databases, data stores and/or distributed resources. The database may take any form and may be a Structured Query Language (SQL) database, not only SQL (noSQL) database management system or other data storage system capable of storing data and being queried to retrieve the stored data.

[0152] To aid clarity and avoid prolixity, the database terminology used herein will predominantly relate to SQL database concepts (e.g., tables, records and fields) but it should be appreciated that such terminology also encompasses equivalent or analogous noSQL concepts or other data structure concepts, thus should not be seen to be limiting unless explicitly claimed as such.

[0153] For example, the investigative module is configured to perform a singular investigative function.

[0154] For example, the investigation system is configured to send at least one investigative module selected from a plurality of investigative modules collectively forming an investigative module library. For example, the investigative module library includes a database in one or more

data stores, the database including one or more related or unrelated records including data relating to the investigative modules. The library may include the investigative modules within records or may provide links in the records to a data store location containing the investigative modules.
[0155] For example, the investigative module library includes at least one investigative module configured to perform a singular investigative function.
[0156] For example, the investigative module library includes at least one investigative module of less than 100 KiloBytes and, in one example, of less than 10 KiloBytes.
[0157] For example, the majority of the investigative modules are small in size, i.e. less than 100 KiloBytes and, in one example, of less than 10 KiloBytes.
[0158] For example, the at least one investigative module is configured to run on the host computer independently from the investigation system.
[0159] For example, the investigation system is configured to send multiple investigative modules to the host computer. The multiple investigative modules may be sent together in single or multiple sessions, streamed, as a single data transfer or separately in multiple data transfers.
[0160] For example, multiple investigative modules may be sent, and the investigative modules configured to run on the host computer simultaneously, sequentially or according to a predetermined scheme. The scheme may be randomised to be unpredictable.
[0161] Reference herein to the term “unpredictable” with respect to a time schedule or selection should be understood to include random and pseudo-random times or time-ranges or any predetermined non-random selection that aims to prevent or at least hinder another system or entity from knowing with certainty the time that the at least one investigative module will be sent and/or run on the host computer.
[0162] For example, the investigation system is configured such that the at least one investigative module is run according to an unpredictable time schedule. This may be achieved in a number of ways, for example: [0163] sending investigative modules to the host computer according to an unpredictable time schedule; [0164] by configuring investigative modules to execute according to an unpredictable time schedule, e.g. an investigative module may be configured to self-execute after a randomised time delay after sending; [0165] by configuring investigative modules to self-execute to perform their investigative function upon detection of an unpredictable trigger, e.g. system entropy level, unpredictable level of network data received and/or sent; unpredictable time, network, CPU, graphics processing unit (GPU), Disk or Memory load, memory load or other system parameter; [0166] by establishing connections to the host computer according to an unpredictable time schedule; [0167] combinations and permutations of the aforementioned.
[0168] For example, the investigation system is configured to generate an unpredictable time schedule, the unpredictable time schedule including a randomly generated execution time for each investigative module, or group thereof.
[0169] For example, the randomly generated execution time is sent to the corresponding host computer as computer readable instructions or data instructing the corresponding host computer to run the corresponding investigative module, or group thereof.
[0170] For example, the investigation system is configured to send the at least one investigative module, or group thereof, to the corresponding host computer at the execution time.
[0171] The unpredictable time schedule may include a time window within which the execution time is specified. Multiple time windows may be provided with corresponding execution times within each window to provide a repeating schedule by which the at least one investigative module may be executed repeatedly.
[0172] The execution time or time window may be modified unpredictably (e.g. manually, pseudo-randomly or randomly) to provide the unpredictable time schedule.
[0173] In one embodiment, additional parameters may be incorporated in a randomiser calculation to determine the execution time and/or time window for running the at least one investigative module. The additional parameters may include: [0174] performance parameters of the host

computer or investigation system e.g. processor, memory or network usage; [0175] system entropy; [0176] external randomiser variable, e.g. atmospheric noise or other randomiser variables; [0177] combinations of the aforementioned; [0178] Any other variable

[0179] For example, the investigation system is configured to send an unpredictable selection of investigative modules such as, for example, selected from the investigative module library.

[0180] Reference herein to the term “unpredictable” with respect to the selection of investigative modules should be understood to include random and pseudo-random selections or any predetermined non-random selection that aims to prevent or at least hinder another system or entity from knowing with certainty which investigative modules will be sent and/or run on the host computer. This unpredictable selection may be achieved in a number of ways, for example: [0181] selecting one or more investigative modules at random using a randomiser function; [0182] including a random selection condition in each investigative module which determines if the investigative module is selected, the selection condition may be based on a system parameter such as: the system entropy level, level of network data received and/or sent; time, network, CPU, GPU, Disk or Memory load, memory load or other system parameter; [0183] including a random selection condition in each investigative module which determines if the investigative module is selected, the selection condition may be based on other system parameters such as geographic or network location, host tags; [0184] external randomiser variable, e.g. atmospheric noise or other randomiser variables. [0185] combinations of the aforementioned; [0186] any other unpredictable technique.

[0187] In one embodiment, at least one investigative module is configured to include an internal time schedule indicating a time(s) to be run on the host computer after being sent.

[0188] For example, the investigation system is configured to provide an unpredictable selection of investigative modules to send to the host computer and/or to be run on the host computer.

[0189] For example, the investigation system is configured to generate or receive a specification of a percentage value indicating a proportion of the selected investigative modules to run.

[0190] In another embodiment, the investigation system is configured to generate or receive a specification of a percentage value indicating a proportion of all available investigative modules to run.

[0191] For example, the investigation system may include a scheduler that may modify parameters to provide an unpredictable schedule of investigative modules to be sent to and/or run on the host computer, the parameters including: [0192] a time window and/or execution time in which to send and/or run the at least one investigative module; [0193] a selection of investigative modules to run; [0194] a percentage value indicating a proportion of investigative modules to run.

[0195] The investigation system is thus configured to perform a non-contiguous investigation by running the investigative modules non-contiguously, i.e. not all of the investigative modules available are sent and used to scan the host computer together as is common for prior art systems, instead the investigative modules may operate independently and are sent individually or in groups as a result of the aforementioned scheduling.

[0196] The non-contiguous investigation thus hinders or counteracts the ability for an intruder to detect the presence of the investigative modules and/or to predict what modules are run and when.

[0197] For example, the investigation system includes at least one control server and at least one scanning node. For example, the at least one scanning node establishes the connection to the host computer and is connectable to the control server. The scanning node may be a computer server.

[0198] The scanning node and control server may use asymmetrical cryptography to store login credentials for the host computer, the scanning node holding a private key and the control server holding a public key. For example, the control server receives and then encrypts login credentials for the host computer using the public key and provides the encrypted login credentials to the scanning node for decryption to establish a connection to the host computer.

[0199] For example, the control server includes the scheduler, defining timing and the investigative

modules to be sent to the host computer.
[0200] For example, each investigative module may perform one or more investigative functions on the host computer to investigate the host computer to ascertain if the host computer has any data forms with suspicious attributes.
[0201] Data form attributes may for example include name, directory, location, size, function, origin, history, activity, execution time, duration, throughput, privilege, or indeed any other attribute of a data or process.
[0202] For example, the host computer is deemed to be in an unsuspicious state when all the host computer data form attributes comply with corresponding predefined integrity status.
[0203] For example, a data form is considered suspicious when at least one data form attribute differs from the predefined integrity status.
[0204] The predefined integrity status may be defined in a variety of ways, according to the preferences of the user, convenience, external or internal security or technical standards, historical benchmarks or comparisons, or any other chosen criteria. As an example, in its simplest form, the host computer may be considered to be in an unsuspicious state when part, or all of, the data form attributes match those of a previous instance (ideally corresponding to a previous successful investigation) i.e. the host computer appears unchanged.
[0205] More specifically, the following list details various data forms of the host computer that may be investigated and examples of suspicious conditions for each data form. The exemplary data forms that are listed below that may be investigated are includes under the broad categories of “File”, “Process”, “User”, “Log”, “Directory”, “Reconnaissance” and “Incident Response”.
File Data Forms
[0206] “File” investigative modules may perform the investigative function of investigating the host computer to ascertain if there are any files stored on the host computer (or stored separately but accessible by the host computer) and that have attributes differing from a predefined integrity status.
[0207] To make such an investigation the investigative module may generate a cryptographic hash of one or more directories on the host computer and compare with generated cryptographic hashes of other directories, the comparison indicating whether the condition is deemed suspicious or not.
[0208] Example suspicious file data form attributes are listed below and include: [0209] files with at least one data form attribute matching a corresponding predefined data form attribute stored in data in the investigative module; [0210] files having at least one data form attribute differing from a corresponding predefined data form attribute stored in data in the investigative module; [0211] files with entropy exceeding a predefined threshold value stored in data in the investigative module; [0212] files containing payloads or backdoors matching predefined signatures or attributes stored in data in the investigative module. [0213] Illegitimate files, being files, system shells or other data with deliberately misleading attributes, designations or characteristics to conceal their true function, location or purpose, e.g. system files in non-standard system locations or files that do not appear to be part of the standard system install but are installed as if they are or files disguised as a different file type, e.g. a packet sniffer renamed so it is not easily identified. [0214] Files known to be suspicious or malicious files. [0215] Files that can provide escalated privilege or other malicious features. [0216] Renamed system shells e.g. system shells that have been renamed to conceal their true functionality such as /bin/false or /sbin/nologin that are really system login shells [0217] Files with attributes matching something they are not such as Hypertext Markup Language (HTML) files that are executable code. [0218] Files that contain malicious or suspicious payloads such as backdoors and other dangerous operations. [0219] Files with very high entropy indicating they are compressed or encrypted to prevent analysis. [0220] Files with unusual attributes set such as being immutable to prevent deletion.
Process Data Forms
[0221] Process investigative modules may perform the investigative function of investigating the

host computer to ascertain if there are any processes on the host computer that have data form attributes differing from a predefined integrity status.
[0222] To make such an investigation, the investigative module may generate a cryptographic hash of all the running processes on the host computer and make a comparison with process names or other process data form attributes, the comparison indicating whether the condition is deemed suspicious or not.
[0223] Example suspicious process data form attributes are listed below and include:
[0224] Processes with at least one data form attribute matching a corresponding predefined data form attribute stored in data in the investigative module. [0225] Processes having at least one data form attribute differing from a corresponding predefined attribute stored in data in the investigative module. [0226] Processes without corresponding running binaries on the disk. [0227] Processes running from locations on the corresponding host computer matching a corresponding predefined location stored in data in the investigative module. [0228] Any changes in a process attribute, such as the name, binary location, username that started it, the user and group owner, command line arguments, network ports operating or similar parameters as can be obtained. [0229] Processes that are running but deleted from a storage memory of the host computer. [0230] Processes that appear to be trying to avoid detection. [0231] Processes that have open network sockets to act as a backdoor or server. [0232] Processes that are running with a suspicious combination of features (such as a network sniffer). [0233] Processes running out of suspicious locations such as /tmp, /dev, or the root user home directory. [0234] Processes with unexpected or suspicious names such as being called just one character, having spaces in unusual locations, or trying to look like or impersonate legitimate processes on the system in other ways. [0235] Processes disguised as legitimate processes on the system. [0236] Processes that may be interfacing with the kernel or operating system in suspicious ways to hide activity or otherwise perform illegitimate operations. [0237] Various other ways that a process may be acting strangely to conceal its activity, presence or purpose. [0238] Processes running from hidden or suspicious directory names. [0239] Processes running from directories from a currently unmounted file system. [0240] Processes running from protected system directories that would not normally allow this kind of activity. [0241] Processes running with a binary that has high entropy indicating is compressed or encrypted to avoid analysis. [0242] Processes running on a host that have never been seen before. [0243] Active anti-forensics and other anti-detection features and methods.
User Data Forms
[0244] User investigative modules may perform the investigative function of investigating the host computer to obtain information for ascertaining if there are any users or user accounts on the host computer that have data form attributes differing from a predefined integrity status.
[0245] Example suspicious user data form attributes are listed below and include: [0246] A user attribute matching a corresponding predefined user data form attribute stored in data in the investigative module. [0247] A user attribute differing to a corresponding predefined user data form attribute stored in data in the investigative module. [0248] A user attribute indicating the user has deleted a corresponding command history or linked their command history to a location. [0249] User login data form attributes matching predefined login data form attributes stored in data in the investigative module. [0250] User authentication tokens with data form attributes matching predefined authentication token data form attributes stored in data in the investigative module. [0251] Any user present in the comparison data fingerprint that is not part of the reference data fingerprint of the host. [0252] Any differences between the comparison data fingerprint and reference data fingerprint that indicate changes to known users, such as changes to the user's home directory, user ID, login shell, password or other attributes. [0253] Any differences between the comparison data fingerprint and reference data fingerprint that indicate a new SSH key or other changes to authentication mechanisms to allow access. [0254] Users belonging to the root user ID group but are not named root. [0255] Users trying to conceal their activity by deleting or linking

their command histories to locations such as/dev/null or similar. [0256] Users trying to conceal their presence by deleting themselves from system/etc/passwd files but still remaining in a local password database. [0257] Users making their home directory/dev/null to conceal activity. [0258] Suspiciously added users trying to masquerade as legitimate system default accounts. [0259] Unusual login patterns and behaviour for users. [0260] Suspicious login times, locations, or concurrent logins from a combination thereof. [0261] Authentication tokens that can allow users to login when a casual review would make it seem like they cannot (e.g. orphaned or malicious SSH authentication keys.)

Directory Data Forms

[0262] Directory investigative modules may perform the investigative function of investigating the host computer to ascertain if there are any directories on the host computer that have data form attributes differing from a predefined integrity status.

[0263] Example suspicious directory data form attributes are listed below and include: [0264] Any differences between the comparison data fingerprint and reference data fingerprint that indicate a new or modified directory. [0265] Any differences between the comparison data fingerprint and reference data fingerprint that indicate changes to directory permissions, owners, links, or other directory attributes. [0266] Illegitimate directories, being directories with deliberately misleading data form attributes, designations or characteristics to conceal their true function, location or purpose, e.g. a directory named as a standard system directory but having different case values set (e.g. /etc vs. /Etc). [0267] Hidden directories under system areas or otherwise incorrect locations, e.g. under such as /bin, /sbin, etc. [0268] Suspiciously named directories, e.g. naming a directory “ . . . ” or “ . . . ” (dot space) can look legitimate to an admin quickly doing a directory listing but are often malicious. [0269] Directories that have an invalid link count. This may be achieved by stealth rootkits that have modified the kernel runtime to not show certain named directories, but are not able to modify the actual filesystem to change link structure [0270] Other directories of known malware, rootkits, or other signs of compromise that indicate something malicious. [0271] Unusual directory activity in system locations or otherwise incorrect locations. [0272] Unusual directory activity under system or user account locations. [0273] Directories from a previously mounted file system that has had a legitimate file system mounted on top of to conceal its location/presence. [0274] Directories that have unusual directory attributes such as being set as immutable.

Log Data Forms

[0275] Log investigative modules may perform the investigative function of investigating the host computer to ascertain if there are any logs on the host computer that have data form attributes differing from a predefined integrity status.

[0276] Example suspicious log data form attributes are listed below and include: [0277] Any differences between the comparison data fingerprint and reference data fingerprint that indicate a user has logged into a host where that user has not done so before. [0278] Any differences between the comparison data fingerprint and reference data fingerprint that indicate unusual login activity during suspicious hours (e.g. middle of the night when the user should be asleep). [0279] Any differences between the comparison data fingerprint and reference data fingerprint that indicate a login from network addresses that have not been seen before for that user. [0280] log file tampering, inconsistencies, or other signs that audit logs have been tampered with or disabled on the host computer. [0281] Audit logs such as wtmp, utmp, btmp, or lastlog that have been zeroed out or erased. [0282] Audit logs such as wtmp, utmp, btmp, or lastlog that have been edited to remove entries. [0283] log file tampering e.g. deleted or zeroed out legitimate system logs that track system messages, audit information, or other log activity. [0284] Inconsistent log file tampering that may have modified one kind of audit log such as wtmp, utmp, or btmp, but did not modify other logs that can be used to cross correlate and show tampering took place. [0285] Injection of false, misleading, or inconsistent information into system logs to avoid detection or to appear as legitimate events. [0286] Other log file evasion, deletion, and editing to cover user or

system misuse.
[0287] A further type of investigative module includes ‘reconnaissance’ investigative modules that investigate general information about the host computer be used by administrators to determine the state of the host computer. Reconnaissance investigative module functions include but are not limited to: [0288] Gathering operating system and hardware information on the host computer. This includes operating system versions, patch levels, kernel versions, software loaded, etc. [0289] Gathering hardware information on the host computer such as CPU, hard disk, memory available, etc. [0290] Gathering network information on the host computer such as listening network ports, network activity, or active servers. [0291] Gathering user information on the host computer, such as logged in users, users with valid accounts, newly added users, default users, user directory information, etc. [0292] Gathering process information on the host computer, such as all running process IDs, process names, CPU usage, memory consumption, process types, process files open, disk activity, scheduled tasks, etc. [0293] Gathering log data on the remote system, such as audit logs, kernel logs, process logs, and other relevant data. [0294] Gathering all open files for running processes. [0295] Gathering all cryptographic hashes of running processes. [0296] Gathering all open network ports for all processes and users. [0297] Gathering all in use network ports and the remote system connected to. [0298] Process tree information which includes full process details such as parent and child processes, binary locations, hashes, directories, effective user ID, memory, CPU and network usage. [0299] All logged in users. [0300] All firewall rules. [0301] All routing information [0302] All logins recorded by the system including remote host information for the connection and user IDs. [0303] All bad logins recorded by the system including remote host information and failed user ID used for the login attempts. [0304] Gathering all open files, named pipes, and other associated information with all running processes. [0305] Gathering memory and kernel structures to search for malicious hooks and other malicious activity. [0306] All scheduled tasks (e.g. crontab and system scheduled tasks).
[0307] A further type of investigative module includes ‘incident Response (IR)’ investigative modules that may investigate the host computer to perform deep investigative analysis or collection of data. Incident Response (IR) investigative modules functions include, but are not limited to: [0308] Detecting listening network servers and services, e.g. by gathering all open network ports by direct query. [0309] Collecting log data to save for forensic analysis and prevent tampering on the host. [0310] Collecting data including active and inactive users on the host computer as well as their activity on the host computer. [0311] Detailed hard drive inspection for suspicious directories, files, and processes. [0312] Collecting a full file system cryptographic profile to spot malicious files across other systems. [0313] Collecting memory snapshots to isolate and analyse actively running malicious programs that are on the system. [0314] Retrieving kernel modules status, process table status, file system data form attributes for all system volumes. [0315] Searching for hidden network capture files, hidden binaries, encrypted files and other similar signs of compromise activity. [0316] Any of the previous investigative module types but modified to do deeper and broader searches for problems and return data so a human operator can analyse the data to identify problems. [0317] Initiating reactive countermeasures.
[0318] For example, at least one investigative module is configured to be inoperable when sent to the host computer. The inoperable investigative module may thereby act as a ‘decoy’ for an intruder who may use time and/or resources attempt to stop, remove or otherwise hinder the investigative module. The term “inoperable” as used herein should be understood to include computer programs that are unable to execute, are executable but perform no operations on the computer or are executable to perform an operation that isn't functional to alter the state of the host computer or datthereon.
[0319] An intruder may attempt to identify, analyse or otherwise interfere with the investigative modules and a number of techniques may be implemented by the present disclosure to mitigate such attempts. These techniques may include one or more of: [0320] At least one of the

investigative modules may be configured to run on the host computer and then be removed from the host computer. For example, at least one of the investigative modules may be configured to self-delete after completion. Alternatively, or in addition, at least one of the investigative modules may be configured to remove another investigative module on the host computer. [0321] At least one investigative module may be a computer program with code that is configured to decrypt, decompress and run on the host computer, subsequently self-deleting when finished running.
[0322] At least one investigative module may be encrypted with a disposable key. A scanning node may generate an encryption key and encrypt the investigative module upon sending to the host computer. The host computer may decrypt the investigative module and delete the key once the investigative module has run, thereby preventing subsequent decryption of the investigative module e.g. by an intruder attempting to analyse the investigative module. [0323] At least one investigative module may be a computer program with code that is obfuscated. [0324] At least one investigative module may be a computer program with code that is compressed.
[0325] The method of claim **1**, wherein the at least one investigative module is configured to generate investigation data including results of the investigative function performed by the at least one investigative module.
[0326] Security of data returned by the investigative modules may also be important and it may be necessary to prevent confidential data from being transmitted from the host computer. Many operators may prefer to receive the results of the investigation and perform further investigation and/or remedial action without involvement of an external entity who would otherwise have access to potentially sensitive or confidential data.
[0327] For example, at least one investigative module is configured provide the investigation data to the host computer, e.g. writing to an electronic on the host computer or sending data to a User Interface (UI) on the host computer.
[0328] In one embodiment, the investigative modules may be configured to send data to a machine learning sub-system on the host computer and/or the investigation system to assist in automatically identifying problems.
[0329] For example, the at least one investigative module is configured to generate alert data and send the alert data to the investigation system if it is stopped, removed prematurely or returns investigation data of unexpected data type or content.
[0330] For example, at least one investigative module is a computer program configured to not transmit data retrieved from the host computer to the investigation system. For example, the investigative modules are configured to return data retrieved from the host computer to an electronic file on the host computer.
[0331] In one embodiment at least one investigative module is a computer program that may return the results of the investigative function to the investigation system without providing data retrieved from the host computer.
[0332] For example, at least one of the investigative modules is configured to request additional investigative modules be sent to the host computer, for example, an initial file scanning investigative module may be sent to perform a first scan of the file system and then if any potential threats are identified, the initial scanning investigative module may transmit data to the investigation system indicating a request that one or more additional investigative modules be sent that may perform additional tasks.
[0333] The performance load on the host computer may thus be minimised by only receiving and running those modules that are required, in contrast to prior art local or agent-based systems that attempt a contiguous investigation or real-time threat defence with a correspondingly high-performance load on the host computer.
[0334] Thus, according to another aspect of the present disclosure there is provided a method of minimising the performance load incurred by an investigation of the host computer using an investigation system as aforementioned, the method including investigating the host computer

using a discontiguous process of running investigative modules.
[0335] In some applications the host computer may be formed from a ‘container’ or include one or more containers running thereon.
[0336] As used herein, the term “container” refers to a stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings etc. Containers aim to isolate the software from the surrounding operating system so as to be able to run application processes within the container as if the containerised applications were operating on their own separate system
[0337] In one embodiment, the container may be at rest, (e.g. with no processes running) or active —with processes running.
[0338] For example, results of at least one investigative module's investigation on the host computer are sent to the control server as investigation data.
[0339] For example, the investigation data, indicating results of at least one investigative module's investigation on the host computer are sent to the control server via the scanning node.
[0340] For example, the control server is configured to enable a user to connect to the control server and request an investigation of a host computer, the user providing login credentials to the control server and a request to perform the investigation.
[0341] For example, the scanning node is configured to deny access to the user if the host computer is connected to the scanning node.
[0342] For example, the login credentials for the host computer are stored temporarily in the scanning node to establish the connection to the host computer and persists only until the connection is terminated. For example the login credentials for the host computer are stored in volatile memory of the scanning node, e.g. the scanning node Random Access Memory (RAM). Thus, if a scanning node is compromised and a non-volatile memory store is copied the login credentials are still inaccessible.
[0343] For example, at least one investigative module is configured to scan the host computer for containers, the investigative module returning a list of containers to the scanning node. The scanning node may then send at least one investigative module to the host computer, the investigative module configured to attach to, and investigate the container.
[0344] For example, the at least one scanning node includes decryption keys for encrypted investigative modules and provides the decryption keys to the encrypted investigative modules for decryption when a connection to the host computer is established and the encrypted investigative modules have been sent to the host computer.
[0345] Reference herein is made to various aspects and embodiments of the present disclosure. For clarity, and to aid prolixity, every possible combination, iteration or permutation of features, aspects and embodiments have not been described explicitly. However, it should be appreciated that the disclosure herein should be interpreted to include any combination, iteration or permutation unless explicitly and specifically excluded.

Description

BRIEF DESCRIPTION OF DRAWINGS
[0346] Further aspects and advantages of the present disclosure will become apparent from the following description which is given by way of example only and with reference to the accompanying drawings in which:
[0347] FIG. **1** shows a high-level schematic diagram of an investigation system according to one embodiment;
[0348] FIG. **2** shows a high-level schematic diagram of a method of investigating host computers using the investigation system of FIG. **1**;

[0349] FIG. **3** shows a further schematic diagram of the general method of operation of the investigation system;
[0350] FIG. **4** shows a high-level schematic diagram of the method of operation of the investigation system utilised with containers;
[0351] FIG. **5** shows a flowchart of the general method of operation of the investigation system;
[0352] FIG. **6** shows a flowchart of the general method of operation of investigating a host computer according to one embodiment;
[0353] FIG. **7** shows a flowchart of the general method of operation of investigative modules on the host computer according to one embodiment;
[0354] FIG. **8** shows a high-level schematic diagram of credential management utilised by the investigation system;
[0355] FIG. **9** shows a high-level schematic diagram of a scheduling system utilised by the investigation system;
[0356] FIG. **10** shows a high-level schematic diagram of a method of investigation results management utilised by the investigation system;
[0357] FIG. **11** shows a high-level schematic diagram of a method of investigation results management utilised by the investigation system;
[0358] FIG. **12** shows a simplified high-level flowchart of a method for obtaining and storing a data fingerprint according to one embodiment;
[0359] FIG. **13** shows a simplified high-level flowchart of a method for investigating a host computer and comparing data fingerprints, according to one preferred embodiment;
[0360] FIG. **14** shows a flow-chart of an SSH Hunter investigative module process according to one embodiment;
[0361] FIG. **15** shows a flow-chart of an investigation system process for processing SSH Hunter investigative data;
[0362] FIG. **16** shows a user interface webpage for displaying investigative data retrieved by the SSH Hunter investigative module; and
[0363] FIG. **17** shows another user interface webpage for displaying investigative data retrieved by the SSH Hunter investigative module.

DETAILED DESCRIPTION

Drawing Reference Table

TABLE-US-00001 1 Investigation system 2 Host computer 3 Investigative modules 4 Results/Investigation Data 5 Control server 6 Scanning node 7 Database 8 Container - active 9 Container - at rest 10 REST API 11 Distributed Messaging Queue 12 Encrypted credentials 13 Decrypted credentials

[0364] Reference will now be made in detail to embodiments of the present disclosure, examples of which are illustrated in the accompanying drawings. While the present disclosure will be discussed in conjunction with the following embodiments, it will be understood that they are not intended to limit the present disclosure to these embodiments alone. On the contrary, the present disclosure covers alternatives, modifications, and equivalents which may be included within the spirit and scope of the present disclosure as described herein and as defined by the appended claims. Furthermore, in the following detailed description of the present disclosure, numerous specific details are set forth in order to provide a thorough understanding of the present disclosure. However, embodiments of the present disclosure may be practiced without these specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail so as not to unnecessarily obscure aspects of the present disclosure.
[0365] Some portions of the descriptions in this document are presented in terms of procedures, logic blocks, processing, protocols and other symbolic representations of operations on data bits within a computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work. In the

present application, a procedure, logic block, process, function, or the like, is a self-consistent sequence of steps or instructions leading to a desired result. Reference herein will also be made to various “algorithms” which should be understood to refer to one or more computer-implemented processes, procedures, functions and/or calculations that are capable of accessing, reading, processing, modifying, creating or otherwise manipulating data.

[0366] The “steps” of each method are those requiring physical manipulations of physical quantities. Usually, although not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, transmitted, combined, compared, and otherwise manipulated in a computer system.

[0367] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present disclosure, discussions utilizing the terms such as “aborting,” “accepting,” “accessing,” “adding,” “adjusting,” “analysing,” “applying,” “assembling,” “assigning,” “balancing,” “blocking,” “calculating,” “capturing,” “combining,” “comparing,” “collecting,” “creating,” “debugging,” “defining,” “delivering,” “depicting,” “detecting,” “determining,” “displaying,” “establishing,” “executing,” “filtering,” “flipping,” “generating,” “grouping,” “hiding,” “identifying,” “initiating,” “investigating,” “interacting,” “modifying,” “monitoring,” “moving,” “outputting,” “performing,” “placing,” “positioning,” “presenting,” “processing,” “programming,” “querying,” “receiving,” “running”, “removing,” “repeating,” “resuming,” “sampling,” “scanning,” “selecting,” “sending,” “simulating,” “sorting,” “storing,” “subtracting,” “suspending,” “tracking,” “transcoding,” “transforming,” “transferring,” “transforming,” “unblocking,” “using,” or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0368] Described herein and as shown in the figures is an “investigation system” (**1**). FIGS. **1** and **2** show high-level schematic diagrams of the investigation system (**1**) in operation with remote host computers (**2**).

[0369] As used herein, the “user” may refer to any user of the system, including operators of investigation system (**1**) or other entities requesting use of the investigation system (**1**).

[0370] The investigation system (**1**) is herein described as investigating host computers (**2**) such as Linux based computer systems including servers, workstations, and Internet of Things (IoT) devices as well as cloud infrastructure and containerized applications. However, this should not be seen to limiting as the investigation system (**1**) can also be adapted to investigate other types of host computer (**2**).

[0371] The investigation system (**1**) uses discrete investigative modules (**3**). Investigative modules (**3**) are computer programs, or data for use by such a computer program. Investigative modules (**3**) include relatively small pieces of code, in comparison to a conventional anti-virus or malware program. Being small in data size provides several advantages, including smaller bandwidth and data storage required.

[0372] The investigative modules (**3**) are sent to and executed on the host computer (**2**). Each investigative module (**3**) performs one or more investigative functions and collectively the investigative modules (**3**) may perform a wide variety of investigative functions.

[0373] The investigative functions are designed to investigate the host computer to ascertain if the host computer has any data or process (hereinafter collectively referred to as data forms) with suspicious data form attributes. The data forms include any data or software, process, manipulation, transmission, operation, deletion or part thereof associated with the host computer, including active, dynamic or static data or processes.

[0374] The host computer (**2**) is deemed to be in an unsuspicious state when all the host computer data form attributes comply with corresponding predefined integrity status.
[0375] Axiomatically, a data form may thus be considered suspicious when at least one data form attribute differs from the predefined integrity status.
[0376] The investigation system (**1**) may thus perform the same or greater number of investigative functions as a comparable system having extensive functionality but written as large computer programs, i.e. the investigation system (**1**) can be operating in a relatively fragmentary manner.
[0377] The investigative modules (**3**) are sent to the host computer (**2**) and run on the host computer to investigate the host computer (**2**) to determine if the host computer (**2**) has been compromised in some way, e.g. if malware or an intruder is present. The investigative modules (**3**) may then optionally send the investigation results (**4**) back to the investigation system (**1**) for further review and analysis.
[0378] The investigation system (**1**) in this embodiment requires the following on a host computer (**2**): [0379] Secure Shell (SSH). [0380] Ability to run an investigative module. [0381] A Unix command shell if investigative modules cannot be executed.
[0382] SSH is used to securely connect to the host computer (**2**) over an encrypted channel. SSH is an industry standard security protocol used to access computer systems. SSH is present on almost all network connected infrastructure hosts by default.
[0383] In some embodiments it may not be possible to push an investigative module (**3**) the remote host computer (**2**) and execute. In these embodiments, the investigation system (**1**) is configured to perform the investigative function using Unix shell scripting as a shim, to collect relevant data, which is then analysed by the backend server (**5**) with a rules engine.
[0384] The investigative module (**3**) pushed to the remote computer is a binary, pre-compiled to run on the remote host computer (**2**). The investigative module (**3**) is agentless, meaning it doesn't need any specialised software on the host computer (**2**) to run. The investigative module (**3**) contains many functions that can instruct the host computer (**2**) to perform various operations, e.g. to retrieve data.
[0385] As the investigation module (**3**) is agentless, it can function on a large variety of host computer types without any kind of modification or software agent install on the host computer (**2**).
[0386] Thus, the agentless nature of the investigation system (**1**) enables it to be deployed for work immediately (without requiring software agent install) even on host computers (**2**) that have never had any kind of security monitoring. All the host computer (**2**) needs are SSH or other access control and the ability to run a binary. Most servers and other computer systems have such abilities as part of their default software packages.
[0387] FIG. **1** shows a high-level schematic diagram of an exemplary investigation system (**1**) structure and includes a control server (**5**) and scanning nodes (**6**) both formed by computer servers with server processor, memory and other typical server computing resources. The control server (**5**) stores or has access to a database (**7**). The scanning nodes (**6**) are typically other servers that are provided to push modules (**3**) and send data from the remote host computers (**2**) to the server (**5**).
[0388] There may be many scanning nodes (**6**) forming part of the investigation system and are an effective way to distribute resources across networks.
[0389] A more detailed description of the investigation system (**1**) is now described. Broadly, the investigation system (**1**) can be deemed to be formed from three main components: [0390] Control Server (**5**) [0391] Scanning Nodes (**6**) [0392] investigative modules (**3**)
[0393] The server (**5**) is responsible for interacting with users over a web interface communicating to a REpresentational State Transfer (REST) application programming interface (API) (**10**). The server (**5**) also interacts with a backend database (**7**) (the system is database agnostic). The server (**5**) also acts as the communication API for scanning nodes (**6**) to receive orders for scans and pass back results.
[0394] The scanning nodes (**6**) communicate with the server (**1**) over a distributed messaging queue

protocol (**11**) such as Advanced Message Queuing Protocol (AMQP) or in other embodiments a direct connection to the server to the REST API over a secured channel such as Transport Layer Security (TLS). The type of protocol is not important as it is predominantly used as a tunnel to send scanning order manifests to be run against host computers (**2**). Order manifests are detailed lists of what host computer (**2**) to scan, investigative modules (**3**) to use for the scan, and encrypted login credentials. The scanning node (**6**) receives these manifests, sends the relevant, investigative modules (**3**) to perform the requested scan, and returns the results from the scan back to the server (**5**) over a REST API (**10**).

[0395] Investigative modules (**3**) are the computer programs (or data used thereby) that are pushed to the remote host computers (**2**) to perform a variety of tasks such as security checks and remediation.

[0396] The investigative modules (**3**) used in this embodiment includes a first investigative module (**3**), which includes the binary as aforementioned and will hereinafter be referred to as the 'binary'. Subsequent investigative modules (**3**) are then sent to the binary. The subsequent modules include small JavaScript Object Notation (JSON) code modules that each provide instructions (data, parameters and/or functions) for an investigative function and are processed by the binary for each investigation. The investigative modules (**3**) are designed to be very small and typically only perform a singular investigative function. The modules can thus be created with minimal data size and can be transmitted across the network easily.

[0397] The investigative function can be instructions for the binary to perform a function or may be a function that the binary can run if the binary doesn't have the necessary function already.

[0398] Once the subsequent investigative modules (**3**) arrive on the host computer (**2**), they are processed by the binary to run the relevant investigative function. The investigative functions will return results in the form of investigation data.

[0399] The binary is written in Go in this embodiment, but other languages may be utilised, e.g. Rust, Java, or C. An investigative module (**3**) can be written in any language if the host computer (**2**) supports it. In a further embodiment the investigative modules use Unix shell scripting if other languages are not available, and no other options are present. This Unix shell embodiment collects similar data and sends it to the backend server (**5**) for analysis.

[0400] The investigative modules (**3**) may collectively form a library of investigative modules (**3**). Investigative modules (**3**) can then be selected from the library to run on the host computer (**2**). The selection can be performed by users, administrators, scanning nodes (**6**) or the control server (**5**) as appropriate for the application.

[0401] New investigative modules (**3**) are added to the library to perform new functions. An advantage of this investigation system (**1**) is that investigative modules (**3**) can be written quickly and deployed without needing to modify the host computers (**2**), update any agent thereon, and typically without even needing to modify the scanning nodes (**6**) or server (**5**). Users can add new investigative modules (**3**) to the library, and they are immediately ready for deployment. Unlike prior art systems that may require major updates to deliver upgraded security capabilities, new investigative modules (**3**) can be added without any modification to the host computers (**2**) or the rest of the investigation system (**1**), and the new modules can be deployed immediately.

[0402] The server (**5**) has a REST API interface (**10**) built to handle calls from a User Interface (UI) and scanning nodes (**6**). The UI provides a graphical front-end to the server (**5**) for ease of use. The REST API (**10**) can also be called by anyone without a UI if it's required to incorporate the investigation system (**1**) into a larger existing product such as a Security Information Enterprise Management (SIEM) tool or other monitoring software.

[0403] The REST API (**10**) also allows scanning nodes (**6**) to authenticate and return data for protected host computers (**2**). The server (**5**) is responsible for the management of host computers, login credentials, and investigative modules (**3**). It is also responsible for scheduling and executing of scans, collecting results, reporting alerts, and displaying results to the user for analysis. The

server (**5**) is also responsible for communicating investigation requests to the scanning nodes (**6**) over the messaging queue (**11**).
[0404] The scanning nodes (**6**) are responsible for logging into host computers (**2**) to execute investigative modules (**3**) and report results (**4**). The scanning nodes (**6**) receive requests for investigations (called orders or order manifests) from the server (**5**). Scanning nodes (**6**) also receive encrypted login credentials to use for gaining access the host computers (**2**).
[0405] The scanning nodes (**6**) are able to run a multi-threaded module to send investigative modules (**3**) to rapidly investigate network systems and report the results to the server (**5**).
[0406] Scanning nodes (**6**) are designed so that they can run on either publicly visible computer systems or on private internal networks. Data collected by the nodes (**6**) in any of these contexts can be sent back to the server (**5**) for analysis. In this way, the investigation system (**1**) can work comfortably not only on publicly facing services such as web servers, but also on private cloud or internal infrastructure without any modifications.
[0407] The investigation system (**1**) does not normally limit the number of scanning nodes (**6**) that may operate. Each scanning node (**6**) connects to the messaging queue (**11**) and can receive, and process requests as required. As scanning nodes (**6**) reach capacity, new scanning nodes (**6**) can be added to take on new workloads. When workloads decrease, scanning nodes (**6**) can then be taken offline without any impact to the operation of the investigation system (**1**). Thus, the investigation system (**1**) can be scaled to handle very large networks across public or private networks.
[0408] As hereinbefore described, investigative modules (**3**) include code designed to be pushed to host computers (**2**) to perform targeted security analysis and response. An advantage of the present investigation system (**1**) is the depth and breadth of investigative module types that can be deployed along with the rapid development and deployment capability. Investigative modules (**3**) are highly focused on a singular or low number of investigative functions to minimise size and resource demand. This minimisation helps to prevent overwhelming network and computer resources.
[0409] In preferred embodiments, at the time of writing, the average investigative module size is around 1-3 KiloBytes (KB) in size and about 20-50 lines of code. This means that many investigative modules (**3**) can be pushed to a host computer rapidly even on a modest network connection. The small size means that the investigative modules (**3**) are easier to write and debug as they are not trying to perform a great number of functions. Investigative modules (**3**) may be used to target particular problems or sent in random swarms to do a dragnet style search for incidents without overwhelming the network or host computer.
[0410] The use of investigative modules (**3**) aids in avoiding the large monolithic code problem many prior art systems have whereby they try to scan for all problems at once with a very large and complicated codebase. Small investigative modules (**3**) offer advantages in that they simplify development, reduce code complexity, enable faster targeted response to threats, and allow the system to have a higher overall performance.
[0411] The Investigative modules (**3**) are also typically written to be generic in how they detect signs of compromise to make them more flexible in deployment. These generic investigative modules (**3**) thus do not rely upon static signatures such as “worm variant A” or “worm variant B” detection. Rather, the investigative modules (**3**) may detect generic signs that something is amiss. In this way the investigative modules (**3**) can spot signs of problems without knowing exactly what the name of the attack is or how the host computer (**2**) was compromised. This generic approach provides advantages because the investigation system (**1**) is potentially able to spot common and uncommon compromises without needing a continual supply of updates, as is common with prior art anti-virus and anti-malware products.
[0412] The investigative modules (**3**) are designed to take an analogous approach to that of a human forensic investigator for what they would search for on the host computer (**2**) to detect if something is wrong. A human investigator will normally look for general signs something is

suspicious instead of trying to detect variant X, Y, or Z of a particular attack. The investigation system (**1**) may emulate this principle as well. Instead of trying to know the exact cause of a system compromise, the investigation system (**1**) may provide the user, administrator or investigator with data that, based on generic or targeted investigation concepts, the host computer is either compromised or not.

[0413] A broad description of the investigation system (**1**) will now be described with respect to its operation on an exemplary host computer (**2**) and with respect to FIGS. **3-7**.

[0414] FIG. **3** shows a high-level schematic of how the investigation system (**1**) scans a host computer (**2**) using a native binary or Unix shell scripting. The process includes: [0415] a) a scanning node (**6**) establishes a connection to the host computer (**2**) with SSH credentials supplied by a user. [0416] b) the scanning node creates a randomly named directory on the remote host computer (**2**), ideally in a secure area on the host computer (**2**). [0417] c) the scanning node (**6**) sends investigative modules (**3**) to investigate the host computer. [0418] d) the investigative modules (**3**) execute on the host computer (**2**) to perform their investigative functions. [0419] e) the results from the investigative modules (**3**) investigations are sent back to the scanning node (**6**) and then to control server (**5**). Results may include relevant forensic data that can be used for further analysis if required. [0420] f) The investigative modules (**3**) are deleted from the host computer (**2**) entirely and no presence is left behind. [0421] g) the randomly named directory on the host computer (**2**) is deleted. [0422] h) The SSH connection is closed down.

[0423] It should be noted that the investigative modules (**3**) can be run on the host computer without any connection to the scanning node (**6**). The connection is only needed to send the modules (**3**) and retrieve the results of the investigation.

[0424] The investigative modules (**3**) may have many functions and may check for various signs of intrusion, indicators of compromise, collect forensic data, run response actions or perform other functions as defined by the investigative module (**3**).

[0425] In addition to the above, the investigative modules (**3**) can be made container aware as illustrated in FIG. **4**. This means that investigative modules (**3**) can run against host computers (**2**) with Docker™, Podman™, Kubernetes™, or similar container management and deployment platforms. In this configuration, the investigative modules (**3**) can check static container images (**9**) for signs of compromise or can attach to running containers (**8**) to do the same.

[0426] The process for investigating running containers (**8**) is similar to that for checking a standard host computer (**2**) with some additional steps and includes: [0427] a) After connecting with SSH, a list of running containers (**8**) is obtained; [0428] b) the running container (**8**) is attached to the scanning node (**6**); [0429] c) binary (**3**) and investigative modules (**3**) are pushed/attached to the container (**8**); [0430] d) investigative modules (**3**) are then run inside the container (**8**) as they would be on any host computer (**2**); [0431] e) results are returned; [0432] f) the investigative modules (**3**) are deleted from the container (**8**); [0433] g) the scanning node detaches from the running container (**8**) or container host operating system; [0434] h) SSH connection is terminated.

[0435] Even if the container is not running SSH, the scanning node (**6**) can connect with shell access or similar mechanism and use investigative modules (**3**) as it does on a host computer (**2**). These investigative modules (**3**) once attached to the executing container (**8**) can then be run and report back findings as normal.

[0436] Docker™ and similar containerized applications at rest (**9**) (e.g. not running) may be present on a disk in the standard file system of a host computer (**2**). investigative modules (**3**) made to look for problems inside containers at rest (**9**) can investigate the filesystem directly and report back their findings as discussed previously.

[0437] Docker™ and similar containerized applications that are executing (e.g. running a micro-service such as webserver or database) can be scanned as well. Although containerized applications offer significant security benefits over those that are not in a container, it is still important to

inspect inside running containers to make sure that they have not been compromised.
[0438] There is a difference between the host computer (**2**) running the containers and the containers themselves though. It is entirely possible that the host computer (**2**) is not compromised while containers it is serving have been compromised. Meaning that the attacker was not able to escape the container (which is by design) to attack the container host computer (**2**), but the container itself is no longer secure.
[0439] Thus, the investigation system (**1**) may enable administrators to look inside containers for actively running threats just as they can the host computer (**2**) running the containers.
[0440] As shown in FIG. **3** the procedure of SSH logging in and pushing over investigative modules (**3**) to the host computer (**2**) is the same as for investigating only the host computer (**2**) as previously described. However, when in a container scanning mode, the investigation system (**1**) can obtain a listing of active containers (**8**) on the host computer (**2**). The investigative modules (**3**) may then attach to these active containers (**8**). Once attached, the investigation system (**1**) can send to the container the relevant investigative modules (**3**) to run. Once the investigative modules (**3**) have visibility inside the container, the investigation system (**1**) will run those investigative functions and report back results just as if those investigative modules (**3**) had run on the host computer (**2**).
[0441] The investigation system (**1**) in this container application uses very small text-based interpreted code for the initial investigative modules (**3**).
[0442] The investigation system (**1**) can work over a standard shell attached to any running container (**8**) or the investigation system can view the container externally from the host OS.
[0443] The file system layers are mounted as per a local disk and scanned by the investigative modules (**3**). Processes inside the container are visible from the host operating system (OS) and can also be inspected by investigative modules (**3**). A containerized process will have a flag set in the results of an investigative module (**3**) to indicate it was seen inside a container to allow further analysis at the server.
[0444] Thus, using the investigation system (**1**) on containers negates the need to move over large binary files which often require special services, which are not typically loaded inside container images.
[0445] Furthermore, most container images do not run an SSH server as standard practice. The investigation system (**1**) can take advantage of this aspect by using SSH to log into the container host computer (**2**), and then once logged in to the main host, using investigative modules (**3**) to investigate the containers the host computer (**2**) is running. This is a very advantageous method to gain visibility into the container runtime without modifying files or processes on the containers.
[0446] In the above method there is no need to install anything on the containerized application other than the first investigative module/binary (**3**) onto the host operating system running the containers.
[0447] The investigation system (**1**) thus requires no software agent inside container images. Even if the container image is not under customer control it is likely that the investigation system (**1**) can scan inside it to look for signs of compromise and alert administrators to a problem.
[0448] FIG. **4** also shows containers at rest (**9**) which are images residing on the host computer storage system. These images (**9**) are used to start running containers (**8**) to serve various services. In this case the scanning node (**6**) does not need to attach to a running container (**8**) but instead can send investigative modules (**3**) to scan container files (**9**) on the host computer (**2**) for signs of compromise using the standard host scanning process described earlier.
[0449] In some embodiments the investigation may be performed entirely on the host computer (**2**) instead of exporting the data to the server (**5**) or elsewhere for analysis.
[0450] Prior art systems may use software agents on the host computer (**2**) or connect to the server (**5**) to conduct comprehensive scanning via remote access. These approaches collect large amounts of data, such as running processes, file hashes, log files, etc. This data is then packaged and sent for

analysis by a remote engine. However, this data may include confidential or sensitive information and sending that data from the host computer (**2**) may be considered a security risk or data breach.
[0451] Therefore, in contrast, the method performed by the investigation system (**1**) is an investigation in place on the host computer (**2**). Investigative modules (**3**) are used that are self-contained to do investigation and analysis on the host computer (**2**) without the need to send any data out to an external analysis engine. This means the existing network architecture can be leveraged to obtain the following benefits.
[0452] Faster investigation—as the host computer (**2**) being investigated is doing the analysis using its own CPU, memory, and disk versus sending volumes of data to a central server for processing over the network. It is much faster to have the host computer (**2**) do an investigation using its own resources than it is to send a large amount of data to a centralized system to process. This is as opposed to many prior art approaches that send large amounts of data over a network to a central system that then needs to analyse the results and may experience network performance bottlenecks. It is difficult to scale such a centralized analysis mechanism as thousands of host computers (**2**) may be connected and all sending back large volumes of data for processing, often simultaneously. The present investigation system (**1**) leverages the distributed computing power of the network to assist in reducing data volumes by performing investigation locally and only sending back data that is determined to warrant further analysis.
[0453] Safer investigation—as potentially confidential data is not sent off the host computer (**2**) to an external system which could expose the data to theft. It presents a risk to send out data from a host computer (**2**) for analysis. If a program is sending large amounts of host computer information for analysis, that data could be useful to attackers if they gain access to the database where it is stored. The data could reveal file locations, user account details, company proprietary data, software installed, and other potentially confidential or sensitive information. Moreover, such prior art systems provide a centralized repository for this data where it wouldn't have existed before. In contrast, the investigation system (**1**) is configured such that it doesn't normally retrieve potentially confidential data unless an investigative module (**3**) determines the confidential data is stored on a compromised system and is required to be retrieved for further analysis. The data may be packaged and sent to the server (**5**) for review by a human and ideally only in the context that the data could be relevant to a compromised system.
[0454] In an alternative embodiment, utilising Unix shell scripting bulk data may be collected and analysed by a backend rules engine with largely similar results to the aforementioned method. In this embodiment the investigation system (**1**) is configured to ensure minimal confidential data is extracted from the host computer (**2**) during the analysis. Only data necessary to determine if a compromise has occurred is collected unless the operator initiates further deeper inspections.
[0455] FIG. **5** shows a flowchart of a general method of operation including various steps in an investigation process and includes: [0456] a) Host SSH credentials are received by server (**5**) from a user. The server (**5**) encrypts the SSH credentials with a scanning node public key and can no longer access the credentials. Credentials are stored in the server database (**7**) encrypted with the node public key. [0457] b) The user can add a host computer (**2**) by IP address, IP address range, IP list, API access to a cloud service provider, or other similar mechanism. They also provide the server (**5**) with the encrypted credentials to use to enable SSH login for the host(s). For instance, if the user added an SSH credential called “web_server_credentials”, the user can instruct the server (**5**) to use those credentials when trying to log into the user's web servers (host computer (**2**)) to perform the investigation. Records are loaded into the database (**7**) with the host credentials received at a), the records containing data including the host computer IP address or other identifier and received credentials; [0458] c) An instruction is received to scan host computers (**2**), either from a manual instruction or as a result of a scheduler indicating a host is to be scanned. After adding at least one SSH credential and valid host computer (**2**), the user can provide requests for system information, security scans on demand, or scheduled scans of the host computer; [0459] d)

An order, including computer-readable data, is created with selected investigated modules (**3**) to be sent and the host computer credentials; [0460] e) The order is sent to the scanning nodes (**6**) via the network; [0461] f) The scanning node (**6**) is programmed to select an order from the messaging queue (**11**); [0462] g) The scanning node (**6**) decrypts the login credentials received in the order. [0463] h) The scanning node (**6**) logs into the remote host computer using SSH with the credentials provided and now decrypted. The scanning node (**6**) then sends encrypted investigative modules (**3**) to the host computer (**2**), including a first investigative module (**3**) in the form of a binary. [0464] i) The investigative modules (**3**) are then decrypted, executed and return results through the REST API (**10**) to the scanning node (**6**), or in alternative embodiments to the host computer (**2**). In execution, the binary uses the other investigative modules (**3**) now present on the host computer (**2**). If the investigative binary is not available, a Unix shell script shim is sent which collects similar data but sends the data back to the control server (**5**) via scanning node (**6**) for rule engine analysis. [0465] j) The investigative modules (**3**), including the binary are then deleted from the remote host computer (**2**). [0466] k) The results received by the scanning node (**6**) are sent back to the server (**5**). [0467] l) The Server (**5**) determines if further processing is needed and sends an order to the scanning node (**6**) if further investigation is required. The order will contain a list of further investigative modules (**3**) to send to the host computer (**2**). [0468] m) If any threats are detected and indicated in the results, details are forwarded as alerts to an operator, file, or user. [0469] n) If any threats are detected and indicated in the results, the threats may be remediated through sending further investigative modules (**3**) via order to scanning node (**6**) or details sent to the host computer for manual remediation or remediation by local system. [0470] o) The entire aforementioned process repeats on a scheduled or manual basis.

[0471] FIG. **6** shows a method of investigating the host computer (**2**) according to one embodiment. The method generally includes the following steps: [0472] a) The control server (**5**) selects the host computer (**2**) to be investigated, either due to manual instruction or according to a schedule. [0473] b) The control server (**5**) runs a scheduler (shown in FIG. **9**) to select the investigative modules (**3**) that are to be sent to and run on the host computer (**2**). Alternatively, a user may manually specify which investigative modules (**3**) are selected. [0474] c) The control server (**5**) queries the database (**7**) with an ID or other unique data form attribute of the host computer (**2**) selected at step a) and then retrieves the host computer information and encrypted login credentials for that host computer (**2**). [0475] d) Similarly, the control server (**5**) queries the database (**7**) with an ID or other unique data form attribute of the investigative modules (**3**) selected at step b) and retrieves investigative module information. [0476] e) an order is created in the form of an electronic file that contains data such as host computer address, the encrypted credentials, and the investigative modules (**3**) to run. [0477] f) the order is sent to the messaging queue (**11**) on the control server (**5**). [0478] g) the scanning node (**6**) queries the messaging queue (**11**) for pending orders from the control server (**5**) in the messaging queue and selects an order. The order selected may be randomised, based on longest pending order or any other condition, to assist with load balancing, or if scanning nodes operate on different network segments and cannot access the requested host to scan. [0479] h) the scanning node (**6**) decrypts the encrypted login credentials using the scanning node's secret key. [0480] i) the scanning node (**6**) reads the order manifest and retrieves the corresponding investigative modules (**3**) from disk. [0481] j) the scanning node (**6**) connects to the host computer (**2**) and logs in with the decrypted credentials. The scanning node (**6**) then sends the investigative modules (**3**) to the host computer (**2**) and they are executed thereon. [0482] k) at least one of the investigative modules (**3**) is encoded with a tag or switch indicating it should be deleted and/or other modules and the directory are to be deleted. The investigative modules (**3**) therefore self-delete after execution and the created directory is deleted, thereby removing all trace of the investigative modules (**3**) on the system. As a backup, the scanning node (**6**) may also send instructions to the host computer (**2**) to delete the investigative modules (**3**). [0483] l) The scanning node (**6**) also deletes the login credentials. [0484] m) as part of step j) the

investigative modules (**3**) send results of their investigative functions back to scanning node (**6**) as data in an electronic file. [0485] n) these results are then sent to the control server (**5**) to initiate further actions as necessary.

[0486] FIG. **7** illustrates the general process and components of the investigation on the host computer (**2**) and includes: [0487] a) The connection of the scanning node (**6**) to the host computer (**2**) by decrypting login credentials and logging into the host computer (**2**). [0488] b) The scanning node (**6**) then sends instructions to the host computer (**2**) to create a file directory (with random name generated by scanning node). [0489] c) The directory is ideally created in a secure area of the host computer (**2**) such as RAM to prevent damaging data on the host computer (**2**). Alternatively, the directory may be created in non-volatile memory. [0490] d) The scanning node sends the investigative modules (**3**) to the directory in RAM. [0491] e) The investigative modules (**3**) are run on the host computer (**2**) and collect data. [0492] f) Errors or alerts are reported to the scanning node (**6**) if an investigative module (**3**) takes too long to execute, is halted or is stopped by the host computer (**2**). [0493] g) The results of the investigative module (**3**) investigation are written to a JSON file then Base64 encoded for transmission of the SSH channel to the scanning node (**6**). [0494] h) At least one of the investigative modules (**3**) is encoded with a tag or switch indicating it should be deleted and/or other modules and the directory are to be deleted. The investigative modules (**3**) therefore self-delete after execution and the created directory is deleted, thereby removing all trace of the investigative modules (**3**) on the system. As a backup, the scanning node (**6**) may also send instructions to the host computer (**2**) to delete the investigative modules (**3**). [0495] i) The scanning node (**6**) sends instructions to the host computer (**2**) to delete the randomly named directory to ensure there is no residual trace on the host computer (**2**) that an investigation has been performed. [0496] j) The scanning node (**6**) deletes the decrypted login credentials to ensure that even if the scanning node is compromised the login credentials are inaccessible. [0497] k) The scanning node (**6**) then awaits a new order and once received repeats the preceding steps for the new order.

[0498] The scanning nodes (**6**) ideally don't communicate with the end-users directly. Scan requests come from the server (**5**) via a manifest over the messaging queue (**11**). Results (**4**) are sent back to the server REST API (**10**) for display and storage. Users cannot log into the scanning nodes (**6**) directly though the server (**5**) or UI.

[0499] The reason for this isolation is due to how the scanning nodes (**6**) login to the host computer (**2**). The investigation system server (**5**) stores encrypted SSH login credentials but cannot decrypt them. The scanning nodes (**6**) contain decryption keys for SSH login credentials but does not store SSH keys itself. A user or intruder needs access to both the control server (**5**) (which holds the encrypted SSH keys) and scanning nodes (**6**) (which can decrypt the SSH keys) to access encrypted SSH credentials.

[0500] This split in responsibility provides extra security in the case either the control server (**5**) or a scanning node (**6**) is compromised. Without having both compromised simultaneously, it makes it harder for an attacker to access SSH credentials for the network. Attackers need the encrypted SSH credentials stored on the investigation system server (**5**) and the decryption key stored on the secured scanning nodes (**6**) to access valid SSH login credentials. By blocking user access to the scanning nodes (**6**) an extra layer of security is provided against having the node private keys taken if there were a vulnerability in the UI. In this case of an investigation system server (**5**) compromise an attacker can gain access only to the database (**7**) but will not have access to scanning nodes (**6**) that contain sensitive decrypted information.

[0501] This investigation system (**1**) contrasts to prior art systems that may store credentials entirely in a single data store. In many prior art systems, a compromise of the data store means that all credentials are compromised.

[0502] Securing SSH keys and username/passwords against theft is important. If an attacker gains access to SSH credentials it means they can access host computers quickly and with little chance of

detection.
[0503] The investigation system (**1**) works to mitigate this potential problem in two ways, Separation of responsibilities and Encryption of credentials at rest.
[0504] As illustrated in FIG. **8**, during installation of the scanning node (**6**) a public/private key pair is generated. The private key is kept secured on the scanning nodes (**6**). The public key is installed on the control server (**5**). The private key must be kept secret on the scanning nodes (**6**). The public key can be made freely available, but for purposes of security it is kept secured by the control server (**5**) as an extra precaution.
[0505] When a user wishes to add SSH credentials to allow the investigation system (**1**) to login and scan a host computer (**2**), the control server (**5**) takes the SSH credentials and encrypts them with the scanning node public key. The encrypted data is then inserted into the database (**7**). The control server (**5**) can no longer decrypt these credentials after this happens. This also means that even if the main control server database (**7**) were to be compromised, no usable host credentials can be stolen and used as they are irreversibly encrypted without the scanning node's private key.
[0506] When a scan is requested, the control server (**5**) collects the host computer information and encrypted SSH credentials and packages it into an order manifest. The order manifest is sent to the scanning nodes (**6**) and contains data specifying what investigative modules (**3**) to run, what host computer (**2**) to run them against, and other relevant information such as how to login to the host computer (**2**) with the SSH credentials.
[0507] The scanning nodes (**6**) take the encrypted SSH credentials and decrypt them with their secret key. If the data decrypts, then the rest of the manifest is run normally. If the scanning node (**6**) cannot decrypt the encrypted data (for instance due to an invalid key), then an error is returned.
[0508] As scanning takes place, the scanning node (**6**) uses the decrypted credentials to login to the host computers (**2**) and collect data. These credentials are not stored on the scanning node disk and only used in RAM. Once the scan is complete, the decrypted credentials are disposed of and the scanning node (**6**) loses access to them until the next scan is sent.
[0509] As shown in FIG. **8**, the server (**5**) stores encrypted credentials (**12**) but cannot read them while the scanning nodes (**6**) can decrypt the credentials (**12**) but does not store them.
[0510] The separation of responsibilities of the scanning node (**6**) and server (**5**) ensures that if the server database (**7**) is compromised that an attacker cannot steal SSH unencrypted credentials to compromise host computers (**2**). At the same time, a scanning node (**6**) that is powered-down does not store SSH credentials used to login. If a scanning node image were stolen for instance, SSH credentials are not able to be accessed as the scanning nodes (**6**) do not store active SSH credentials.
[0511] This arrangement enables login credentials to be changed at only one place at any given time. While other management systems may have SSH keys scattered and make management difficult, the present investigation system (**1**) has SSH credentials stored in one encrypted location on the server database (**7**) where they can be quickly deleted or changed but which cannot be decrypted without the scanning node keys.
[0512] If the user wants to use new SSH keys for login they may delete the old ones from the investigation system server (**5**), add new SSH credentials, and they are again encrypted and ready to be used by scanning nodes (**6**) as part of any new manifests pushed out. During the next scan, the server (**5**) may push down the newly encrypted credentials and the nodes (**6**) can decrypt them as usual.
[0513] Likewise, if a user suspects a node image was stolen, they can delete all credentials, generate new pubic/private keys for the nodes (**6**), and insert new SSH credentials. Thus, the nodes (**6**) can no longer decrypt the SSH credentials and use them maliciously even if powered up by an attacker who gains direct access to the messaging queue system (**11**). The stolen node image has invalid secret keys and they cannot be used to read any credentials encrypted by the changed keys from the server (**5**).

[0514] Adding a host computer (**2**) to be investigated by the investigation system (**1**) requires a valid network address or hostname and login credentials. The aforementioned embodiments use SSH to login to host computers (**2**) so the login credentials need to be either a username/password combination or a generated SSH private/public key pair.
[0515] To add a host computer (**2**), the user may enter either a single IP address, IP address network range, a list of IP addresses, cloud service API access key, or similar method. The user then specifies what credentials to use (as added under Credentials Management). For instance, if the user wants to add all host computers in the development department, they would enter in the IP address range to add. Then the user may specify what credential to use. For example, they may have entered an SSH key pair called “development” for use on all systems that are in the development department of their company.
[0516] After this, the user may provide instructions that all host computers in the specified range are to be added using the specified credentials. The investigation system (**1**) will attempt to login to each host address given with the credentials “development.” If the login is successful, system information about operating system type, versions, etc. will be gathered and returned to the server for database storage. The host computer (**2**) is now considered active and can be added to the scanning pool for future investigation.
[0517] When the investigation system (**1**) scans a host computer (**2**) it will connect using SSH and a user supplied username/password or SSH private key as security procedures require. Once connected, the investigation system (**1**) will then determine what version of the operating system is loaded and various operating system data form attributes such as software versions, build versions, memory size, disk size, kernel versions, and other factors. These data form attributes are then returned to the database (**7**) to be stored with scanned system details.
[0518] If the investigative binary is not able to be loaded, then alternatively a Unix script version of the investigative modules (**3**) can be used. This version leverages the local computer's Unix shell scripting to emulate many data form attributes of the investigative binary. The primary difference being that it relies on bulk data collection of relevant system attributes such as the process tables, directories, and files. This bulk data is then sent back to the server for analysis in the back-end rules engine vs. on the host with the native implementation.
[0519] Once the operating system data form attributes of the host computer (**2**) are gathered, the investigation system (**1**) will then check compatibility for scanning. The investigation system (**1**) may then send various investigative modules (**3**) which may look for signs of compromise or suspected problems on the host computer (**2**). It may also send investigative modules (**3**) that perform remediation or other actions on the host computer (**2**).
[0520] Investigative modules (**3**) are sent to the host computer (**2**) using the SSH Secure File Transfer Protocol (SFTP) protocol. This enables the investigation system (**1**) to send a large number of investigative modules (**3**) quickly. Once sent with SFTP, each investigative module (**3**) is executed on the host computer (**2**) by calling the investigative binary that was pushed to the remote system. If these methods are not available, then the Unix script modules are used to collect bulk data for analysis on the control server (**5**) or scanning node in a rule engine.
[0521] Each investigative module (**3**) executes and collects relevant data to determine the security state of the host computer (**2**) for that module's investigative function. For instance, an investigative module (**3**) that has the function to look for suspicious user entries will look for that single problem and report back its findings.
[0522] Once the investigative modules (**3**) finish running, the collected data is sent back to the investigation system server (**5**) by the scanning node (**6**) for storage in a suitable database for reporting and analysis if needed. This analysis may also instigate follow-on investigations with other investigative modules (**3**) as deemed necessary. The findings for all investigative modules (**3**) are in a standardized JSON format that is encoded for transmission back to the scanning node (**6**) across the open SSH channel for security and privacy. This JSON format is cross platform

compatible and can be expanded as required to report back on any kind of data on the host computer (**2**). Data collected by investigative modules (**3**) can be about system configuration, compromise status, forensic data, or other kinds of information as required.
[0523] The scanning and reporting process may constantly occur across a network. It can occur on an ad hoc, scheduled, or unpredictable basis. The investigation system (**1**) defaults to using a random scheduling mechanism as described below so that the timing of the investigation by the investigative modules (**3**) is unpredictable. Random scheduling offers advantages in making it more difficult for intruders or malware to evade detection. By using a random schedule, it is harder for attackers to devise ways to avoid detection by removing their software ahead of regularly scheduled checks as happens in prior art systems. Furthermore, system performance impacts can be limited by not only using random investigation times, but also running only a selection of all the available investigative modules (**3**) at a time thereby lowering system resource usage. This selection of investigative modules (**3**) may also be random.
[0524] The investigation system (**1**) is configured to give the user the ability to randomly schedule investigative modules (**3**) to be sent to host computers (**2**). As the investigation system (**1**) operates agentlessly and removes all traces upon completion, random scheduling can be used to make intruders or malware evasion more difficult.
[0525] The investigation system (**1**) includes a scheduler as illustrated in FIG. **9** with three primary values/parameters that the User may specify in their request for an investigation of the host computer (**2**). These variables may be modified when scheduling scans of host computers (**2**):
[0526] A time window in minutes (e.g. 60-120). The time window represents a time period in which the investigation and selected investigative modules (**3**) are to be run. The investigation may be repeated in subsequent time windows. [0527] The investigative modules (**3**) to run (e.g. process investigative modules, file investigative modules, individual selection of investigative modules, etc.) [0528] A proportion of investigative modules to run (e.g. 25%)
[0529] Once the user has specified these options they are stored in database (**7**).
[0530] One or more of these values may be made unpredictable to minimize the chance of an intruder being able to escape detection or detect the presence of the investigative modules (**3**).
[0531] For example, the time window may not follow a set time of day pattern. Instead of specifying “run a scan at 6 AM every morning” as prior art systems might, the time window allows a user to select an upper and lower limit. For instance, a user can specify “run a scan every 60-120 minutes with a random scan time between those values.”
[0532] Using the above 60-120 minutes value, the investigation system (**1**) will select a random time to execute the next scan that is equal to or more than 60 minutes and less than or equal to 120 minutes.
[0533] For instance, the value of 73 minutes may be selected as the first random value between 60-120 minutes. When 73 minutes has elapsed, the investigation system (**1**) will run the scan against the specified host computer (**2**). After the scan is run, the scheduler then selects a new random time between 60-120 minutes for the next scan. In this case it may select 95 minutes as the next target time. Meaning that the next scan will happen 95 minutes into the future and so on. This process repeats each time a scan is run to select the next random time a scan is executed.
[0534] A selection of investigative modules (**3**) to run may be made from the library of available investigative modules (**3**). This selection may be predefined, random or determined by the user. The selection may for example include: [0535] a particular group of investigative module types to run (such as only look for file or process attacks); [0536] a custom group, or [0537] all modules.
[0538] The user may also set a random percentage of the investigative modules (**3**) to run for the scan. For example, if a user sets 25%, then the system will randomly select 25% of the total investigative modules (**3**) and then run those at the determined time. Any value between 1-100% can be used. The reason for using percentages less than 100% is twofold: [0539] It reduces system resource impacts on the host computer (**2**) as not all the investigative modules (**3**) are run at once

and instead spreads out the running of investigative modules (**3**) over time and therefore spreads the performance load on the host computer (**2**). [0540] I) It makes evasion by malware or intruders (used interchangeably with the term “attackers”) more difficult. Attackers trying to evade detection do not know what investigative modules (**3**) will be running at any one time. investigative modules (**3**) may be a random set and not a pre-determined list of checks happening on a predictable basis as in the case of prior art systems.

[0541] Thus, not only does the investigation system (**1**) have reduced risk of evasion but it also lowers system performance impact relative to a scan using all available investigative modules (**3**). In one application the scans may collectively include all investigative modules (**3**) being run but spread over a day thereby providing 100% coverage of all investigative modules (**3**) spread out randomly to balance system impact with security. For example, 25% of all investigative modules (**3**) may be run at any one scheduled time. Yet, if that scheduled time happens four or more times a day then there is a probability that 100% of all investigative modules (**3**) will be run. This means the host computer (**2**) can receive greater than 100% coverage daily with investigative modules (**3**) without the massive impacts of full system scans happening on more regular schedules.

[0542] In contrast, prior art scanning solutions load a host computer with large numbers of checks at pre-determined times or on demand. The most typical case is anti-virus systems that are well known for slowing down host computers while they do comprehensive checks for system problems. This is annoying for end users and drastically affects system performance. The present investigation system (**1**) instead operates with minimal performance impact using its random scheduling system and random investigative modules scheduling while also increasing security by making evasion harder for attackers.

[0543] Each investigative module (**3**) by itself is highly focused and may miss attacks it was not designed to see alone. Yet when you have hundreds of investigative modules (**3**) running on a host computer (**2**) each looking for a particular problem, the overall level of protection is very thorough. Moreover, because each individual check is fast and has a low performance impact, the investigation system (**1**) not only provides better protection but does it for a much lower cost in terms of system performance impacts.

[0544] As shown in FIG. **9**, the scheduler is a computer program that includes instructions to query the database (**7**) for a scan to run and retrieves the corresponding execution time. The scheduler includes instructions to run the scan at the scheduled execution time.

[0545] When the time triggers the scan the scheduler queries the database (**7**) for the investigative module (**3**) types to run (as specified by user) and selects a random selection of investigative modules (**3**) with the proportion of user specified investigative modules (**3**) selected matching the random percentage of modules to run.

[0546] The scheduler then updates the database with a new execution time to run the next scan based on the time window specified by the user.

[0547] FIG. **10** shows one embodiment of the processing of results returned by the investigative modules (**3**) and includes the processes: [0548] a) the results are returned from the investigative modules (**3**) to the control server (**5**) via scanning nodes (**6**). [0549] b) the control server (**5**) is programmed to process the data and determine if the results need rule engine processing. [0550] c) If the control server (**5**) determines further processing is required (“Yes”) at step b) the results are sent to a rule engine for further analysis. [0551] d) A rule engine on the control server (**5**) performs further processing of the results to determine if alerts are generated. [0552] e) If the control server (**5**) determines no further processing is required “No” at step b) the results are stored in database (**7**). [0553] f) The control server (**5**) queries the results in database (**7**) and determines if any alerts are found. [0554] g) If no alerts are found, the control server (**5**) logs a ‘pass’ result to the database (**7**) for audit purposes. [0555] h) If alerts are found, e.g. indicating a potential threat, the control server (**5**) sends an alert with relevant information to a User Interface (UI) on the control server (**5**). [0556] i) Alternatively, or in addition to step h) the alert and relevant information may be sent to

email or other alert mechanism. [0557] j) the control server (**5**) may initiate remedial action or other response to the alerts as required. This response is detailed further in FIG. **11**.
[0558] FIG. **11** shows a flowchart of one embodiment of potential response processing to alerts being generated as a result of the process shown in FIG. **10** and includes the control server (**5**):
[0559] generating an alert via some form of alert management system which may include: [0560] a UI alert for the User; [0561] Email message to the User [0562] Chat channel message [0563] any other action communicating the alert to a User or system; [0564] a) determining if remediation is required. [0565] b) if remediation is required appropriate action may be made by sending instructions to the host computer (**2**) or administrator of the host computer (**2**) to, for example, halt a malicious process, quarantine a suspicious file, disable a user account, isolate the host computer (**2**) from network, a user defined action or any other action. [0566] c) if remediation is not required or a compatible remediation action is unavailable the alert is stored in database or sent to the administrator of the host computer (**2**) for manual action.
[0567] The investigative modules (**3**) are standardized to return an easily parsed and cross-system compatible JSON object. JSON is easily indexed and stored in the investigation system (**1**) using any suitable database.
[0568] The investigation system can use any suitable database on the backend to make searching and aggregating data fast and easy. Any style of database searching and indexing could be used that can provide such functionality. The benefit of using JSON format for storing the investigation data is that it is not only easily parsed but is also suitable for use with AI and Machine Learning systems that work well with loosely structured data in a way that traditional SQL systems cannot.
[0569] The investigation system (**1**) may collect an abundance of data from the host computers (**2**) it is protecting. Each scanning node (**6**) can be set to store all data collected, or just store data from failed security checks. The investigation system (**1**) can also store all data that may be required for compliance and auditing reasons. Or storage costs can be reduced by only holding onto data that is critical and shows a problem exists.
[0570] By collecting forensic data and storing it in database (**7**) for indexing and searching, the investigation system (**1**) can protect critical evidence data from tampering or removal on affected host computers (**2**). Collecting and storing data off the host computer (**2**) is important, as one of the primary tasks attackers do when they compromise a host computer (**2**) is to immediately remove their presence from log files and other audit trail to avoid detection.
[0571] When an investigative module (**3**) returns data indicating a problem the data is presented as part of a User Interface (UI). The data may include key pieces of information such as: [0572] Hostname [0573] Timestamp [0574] Relevant network addresses [0575] Process IDs [0576] User IDs [0577] Other relevant data
[0578] Other key data related to the detected problem are gathered. This includes, but is not limited to: [0579] Filenames and file data form attributes [0580] Process data [0581] Cryptographic hashes [0582] Permissions [0583] Environment variables [0584] What was found that was suspicious [0585] Why it was suspicious [0586] What and administrator can do to validate the problem and take action
[0587] This data may be shown in the UI within a browsing interface or can be accessed by third-party products directly through the REST API (**10**). Additionally, the investigation system (**1**) incorporates protection in the results against tampering by providing mechanisms for signing of collected data to show it has not been altered after it was retrieved.
[0588] The investigation system (**1**) is designed to work in a hostile environment and thus as state of a host computer (**2**) is unknown prior to investigation, it is assumed that there could be hostile actors present and that they may try to evade or tamper with security software trying to detect them.
[0589] The investigation system (**1**) addresses these threats in a number of defensive ways.
A—Encrypting and Self-Decrypting Modules
[0590] investigative modules (**3**) may have anti-tamper functionality such that when sent to the

remote host the investigative modules (**3**) are obfuscated, compressed, and then encrypted with a disposable key.

[0591] When executed, investigative modules (**3**) may decrypt and decompress themselves automatically. They will then run their investigative functions and delete themselves. Unprotected investigative module code and decryption passwords are ideally never stored on the host computer (**2**) disk. Even if the investigative module (**3**) is decrypted by a determined attacker, the code is still obfuscated to make reverse engineering a more involved task.

[0592] The above process ensures that if any investigative modules (**3**) are left on the host computer (**2**) (such as from a network timeout or other problem) they cannot be easily analysed. The SSH session storing the decryption keys no longer exists and the investigative modules (**3**) remaining on the host computer (**2**) cannot be easily read without the encryption key and then trying to de-obfuscate the code.

[0593] Encryption and obfuscation prevents attackers from understanding how each investigative module (**3**) works, what it is looking for, and how an attacker could modify their procedures to avoid detection.

B—Appear Unprotected

[0594] The lack of a software agent or any residual modules, directories or the like, means that the host computer (**2**) appears to be unprotected. An attacker entering a host computer (**2**) and doing a cursory scan for active security software is simply unable to detect any presence that the investigation system (**1**) is guarding the host computer (**2**). The presence of SSH is not enough to give away the investigation system's (**1**) presence as these tools are so common as to be above suspicion.

[0595] Some malware is designed to look for signs of security software present and hide, disable, or otherwise bypass it. However, as the investigative modules (**3**) are removed after running there are no traces left on disk that the host computer (**2**) has been investigated. Thus, this type of malware behaviour is ineffectual.

C—Randomness

[0596] As aforementioned, using an unpredictable schedule avoiding fixed times for security checks means that attackers have a harder task designing malware that can uninstall itself at predetermined times to avoid detection by the investigation system (**1**). Furthermore, the ‘swarming’ nature of the investigation system (**1**) sending only some investigative modules (**3**) at a time means that if a problem is found, additional investigative modules (**3**) can quickly be sent on demand to do a more in-depth search for problems. The slightest misstep by an attacker brings about an overwhelming response that makes evasion that much harder.

[0597] The investigation system (**1**) is architected to use scalable distributed scanning nodes (**6**) and command and control infrastructure that can work across public and/or private networks. This architecture allows the investigation system (**1**) to offer protection not only to conventional network servers, but also to provide protection to containerized applications (**8**, **9**) and their host computers (**2**) in cloud environments.

[0598] The investigation system (**1**) also uses scheduling and security scanning methods that limits performance impacts on host computers (**2**). This method allows the investigation system (**1**) to work on a wide variety of systems providing comprehensive protection with a reduced impact on host computer resources.

[0599] The investigation system (**1**) can investigate the host computer (**2**) without requiring a specific software agent on the host computer (**2**). The investigation system may thus be deemed an ‘agentless’ investigation system.

[0600] Software agents require regular updates and can cause conflicts with other system updates that need to be installed. Agents may also use a persistent proportion of host computer resources, e.g. CPU, memory, GPU and other system resources e.g. an agent that requires a conservative average of 5% CPU time means customers need to operate 5% more servers due to this overhead.

This is a significant cost for a large enterprise for no actual gain in security.
[0601] Furthermore, when software agents need to be updated, system administrators must do so against every single host computer (**2**) that the agent is installed on. This process is fraught with problems and requires extensive testing to be sure the updates will not cause conflicts or failures with other loaded software. In addition to this, there is considerable risk doing automated deployment and upgrade of software agents across a network. Any problem caused by the agent or upgrade will impact countless host computers (**2**) and may cause serious disruptions to network operations.
[0602] In contrast, the investigation system's (**1**) agentless architecture does not require constant updates across many systems as new or updated investigative modules (**3**) need only be loaded to the investigation system server (**5**). If new updates are needed or new investigative modules (**3**) created, they are simply placed into the investigation system (**1**) and they can be used without the burden of upgrading hundreds, or thousands of host computers (**2**). Also, as the investigation system (**1**) is agentless it does not require a permanent presence on every host computer (**2**) which means no resources are being used constantly as you′d find with software agents.
[0603] The investigation system (**1**) also sends very small investigative modules (**3**) operable without an agent present on the host computer (**2**). These investigative modules (**3**) can perform investigations very quickly with low system impact as opposed to large monolithic scans found in the prior art. The use of small investigative modules (**3**) combined with the random scheduling nature of the investigation system (**1**) may thus provide superior real-time investigation of a host computer (**2**) with lower performance overhead.
[0604] The agentless nature of the investigation system (**1**) enables it to be used in a rapid deployment or emergency application to quickly detect, isolate, and contain computer system compromises without needing to load software agents onto host computers (**2**). In an emergency, being agentless is important because loading software can alter the forensic integrity of the compromised host computer (**2**). It can also alert attackers or activate anti-forensic features of malware which could trigger destructive responses as the attacker or malware tries to evade detection.
[0605] Additionally, being agentless means that the investigation system (**1**) can secure containerized applications using Docker™ or similar platforms. Since there is no need to load an agent inside the container build specification, the investigation system (**1**) can provide security even to containers being distributed that have no built-in security protections by default.
[0606] The agentless nature of the investigation system (**1**) enables it to be run when needed and then disappear from the host computer (**2**) when completed. For the investigation system (**1**), this feature provides anti-evasion and tamper resistance which increases security. Attackers can evade agent-based and regularly scheduled security checks if they know what is installed and when checks will happen.
[0607] Being agentless, the investigation system (**1**) can show up to a host computer (**2**) unannounced and perform random security checks that can take an attacker by surprise. Because nothing is loaded on the host computer (**2**) there is no indication it is being protected and this makes it harder for attackers to conceal their activity. Further, with no code present on the host computer (**2**) the product cannot be directly targeted for bypass and evasion by tampering as no software is left on protected hosts to invite attack.
[0608] Because it is agentless, the investigation system can be rapidly deployed without costly and time-consuming software installs. This ability to be rapidly deployed means that anyone from the smallest to the largest network would be a user of the investigation system (**1**). Since there is no agent to install, the investigation system (**1**) can quickly be deployed with lower risk to monitored host computers (**2**).
[0609] As discussed earlier, the agentless nature of the investigation system (**1**) means it can run on host computers (**2**) that serve containerized applications (such as Docker™ or Kubernetes™) as

well. The agentless approach means that application containers can be checked either when they are running or at rest on the host computer (**2**) for signs of compromise. Because the investigation system (**1**) does not require an agent to run on each container to be protected, it means that we can assess containers for signs of compromise without impacting the services these containers provide.
[0610] Many Internet of Thing (IoT) devices run embedded versions of Linux that are fully capable of running the investigative modules (**3**). As many IoT devices have limited RAM and CPU power they are not able to run full-time security agents. An agentless solution like the aforementioned investigation system (**1**) is a perfect candidate to provide security to these platforms as it is low impact and does not require any significant system resources to be constantly used.
[0611] Malware or system compromises can be difficult to detect, particularly if the compromise doesn't present a known signature or indicator that can be detected by the investigative modules (**3**). Therefore, it's possible for a host computer (**2**) to be compromised but not have any files or processes that match known threats.
[0612] Existing unknown threat detection relies on either investigation by a person or applying automated behaviour analysis of files and processes, often using learning algorithms. However, such methods are still vulnerable to failure and often return many false positives.
[0613] Many computer systems operate with a predictable, stable set of software and configurations and change very little. Thus, where there is a deviation from such a ‘system reference data fingerprint’ it may indicate a security compromise of the system, accidental or malicious modification, or other events that system administrators, security teams, or compliance teams may benefit from being notified for further investigation.
[0614] Monitoring systems for changes is a known computer security technique and may be referred to as ‘drift detection’. Prior art systems monitor file systems for changes and alert or remediate if they detect a change. However, this often results in false positives as file changes are not necessarily the result of malware or security compromise.
[0615] The present disclosure overcomes these problems by providing a more configurable and contextualised method for drift detection.
[0616] The present drift-detection method involves use of the investigative modules (**3**) to perform investigations of a host computer as previously mentioned and return investigation data.
[0617] In addition to the investigation data, specified modules generate data fingerprints representing multiple data form attributes of the investigation data.
[0618] The investigations are performed on a reference host computer to generate reference data fingerprints. The same investigations can then be performed on a target host computer to generate corresponding ‘comparison’ data fingerprints. The reference and comparison data fingerprints can then be compared, and if they are not identical, there must be a difference in at least one of the data form attributes. Therefore, by performing the same investigations and comparing data fingerprints, drift can be detected.
[0619] In most applications the reference host computer and the target host computer are the same computer and the drift-detection involves repeating the computer investigations over time to generate time-delimited data fingerprints that can then be compared. In other embodiments, the reference host computer may be a computer in a state that is deemed an acceptable reference point.
[0620] For clarity and brevity, the embodiments described herein are made with respect to the reference and target host computers being the same computer, unless stated otherwise.
[0621] FIG. **12** shows a simplified high-level flowchart of the process for obtaining and storing a data fingerprint or ‘match hash’ for a host computer. The method includes: [0622] a) the investigation system performing a computer investigation of a host computer, the computer investigation including: [0623] the investigation system (**1**) establishing a connection to the host computer, and [0624] the investigation system (**1**) sending an investigative module (**3**) to the host computer (**2**), the investigative module (**3**) performing an investigative function and returning investigation data (**4**), the investigation data (**4**) includes multiple data form attributes; [0625] b)

the investigative module (**3**) returning the investigation data (**4**); [0626] c) the investigative module (**3**) generating a data fingerprint by generating a cryptographic hash of the collation of data form attributes of the reference investigation data; [0627] d) the investigation system (**1**) receiving and storing the data fingerprint in a database.
[0628] The method of FIG. **12** is used for generating both the reference and comparison data fingerprints. In preferred embodiments both the reference and comparison data fingerprints are stored in the database to provide users with the ability to make comparisons between different time-delimited pairs of data fingerprints. However, in some embodiments only the reference data fingerprint is stored in the database, the comparison data fingerprint may be processed and a comparison made without requiring the comparison fingerprint to be stored in the database.
[0629] Data form attributes are properties, values or descriptors of a data form, e.g. a file data form may have data form attributes that include the values of filename, content, timestamp, user id, group id, size, mode and path. A hash of a file's content can thus be considered a single data form attribute.
[0630] A cryptographic hash can be generated of the collation of data form attributes using any suitable algorithm and hashing function. In the embodiments shown, the SHA-512 algorithm is used with a hashing function to generate a hash of selected attributes to be compared. For example, a user may wish to detect changes in file attributes such as file name, size, owner, permissions, integrity hash, and creation date. The attributes are concatenated before hashing, e.g. a file with name “abc.js”, size 42 kb and path “root” is concatenated into “abc42root” and then hashed.
[0631] The attribute values may need to be stringified before the hash can be generated if they are in a binary form or for attributes where the binary value of the attribute is used directly (e.g. for generating a cryptographic hash of a running process.
[0632] In another embodiment, the attributes that are hashed may be in the form of an object, the object including key-value pairs of each data form attribute. Where each key is the name or other identifier of the data form attribute (e.g. “filename”) and each value is the corresponding value retrieved (e.g. “abc123.xml”)
[0633] The method first generates a reference data fingerprint of multiple attributes in the investigation data. This reference data fingerprint can be used as a reference for future comparisons to thereby perform drift detection.
[0634] The same investigation is repeated, and a comparison data fingerprint generated. The time-delimited reference and comparison data fingerprints can then be compared to determine if they match. If the fingerprints match, this indicates that there has not been any change in the retrieved data form attributes that collectively form the data fingerprints.
[0635] A reference data fingerprint of the host computer (**2**) is ideally generated in a baseline state, e.g. in a state without known compromise, and that reference data fingerprint therefore represents the corresponding host computer data form attributes in that baseline state.
[0636] Reference data fingerprints need to be generated for all the host computer data form attributes that may be of interest to the user, so that subsequent investigation's data fingerprints can be compared with the reference data fingerprints.
[0637] Many host computers (**2**) could have identical baseline states and therefore should have the same reference data fingerprints. This is common on embedded computers, IoT devices, containers and replicated servers and system images. In such cases, the reference data fingerprints can be generated from a known good host and then used for any other devices that should have the same reference data fingerprint. In this way, corruption, modification, or other changes between systems that should share the same data attributes can also be detected using the present disclosure. This allows users to rapidly investigate many systems and quickly identify any systems that have changed from baseline, thereby shortening the time of the investigation and allowing faster isolation of compromised systems.
[0638] Each data fingerprint is generated by the corresponding investigative module (**3**). Multiple

fingerprints are ideally generated for each investigation, each fingerprint representing a different set of data form attributes from the investigation data.
[0639] The data form attributes in each set are determined by the investigative module (**3**) which specifies which data form attributes are to be collated and hashed.
[0640] A data fingerprint's set of attributes may be a subset of another data fingerprint's set of attributes, providing the user with the ability to detect changes at a subset or superset of data attributes. These multiple data fingerprints can be useful in triggering alerts to different changes, for example, a user may choose to monitor changes at a ‘permissive’ level or at a ‘moderate’ level, representing different sets of data attributes. The data fingerprint for the permissive level may for example include a hash of the combination of the name of a process running on the host computer, and a path for that process. The data fingerprint for the moderate level may include a hash of the combination of the process name, path and SHA-512 hash of the running executable. Thus, for example, a user can choose to look for changes only in the name or path or also detect changes in the running executable.
[0641] A user may wish to allow different versions of the same executable to run without alerting and so choose to be alerted only at the permissive level. Alternatively, the user can select to be alerted at the moderate level, where any attempt to attack a system by replacing an existing, expected binary with a modified copy would generate a drift alert.
[0642] FIG. **13** shows a flowchart of one embodiment of a method for drift-detection of a host computer that is performed when the investigation system (**1**) performs an investigation of the target host computer. [0643] a) In FIG. **13**, the first step a) represents an investigative module (**3**) being run to perform an investigation and return investigation data (**4**). [0644] b) The investigative module (**3**) generates a cryptographic hash of multiple data form attributes of the investigation data (**4**). This cryptographic hash is referred to as the comparison data fingerprint for those data form attributes. [0645] c) The comparison data fingerprint is sent to the investigation system (**1**) for processing. Processing involves retrieving data about the comparison fingerprint, including identifiers of the host computer and the investigative module(s) (**3**) run. [0646] d) A query of the database is run to determine if there is a matching reference data fingerprint for that investigative module (**3**) and host computer, and that is not marked to ignore. [0647] e) If there is no matching reference data fingerprint, then there is no comparison required, and the investigation system (**1**) determines at j) if there is any more data to process. [0648] f) If a matching reference data fingerprint is retrieved from the database, then a data fingerprint comparison is made, comparing the comparison data fingerprint received and the reference data fingerprint. [0649] g) There is then a determination if a match has occurred. [0650] h) If the comparison data fingerprint matches the reference data fingerprint, then this indicates no change/drift and the system processes the next data fingerprint received, repeating c) and d). [0651] i) If the comparison data fingerprint does not match the reference data fingerprint, then this indicates a change/drift and the system produces an alert. The system (**1**) then processes the next data fingerprint received, repeating c) and d). [0652] j) if there are more data fingerprints to process the system, repeats from step c). If there are no more data fingerprints to process then the method is complete and processing ends.
[0653] Users can control the scope of drift detection by choosing the reference data fingerprints that are stored in the database or selecting which reference data fingerprints are monitored e.g. by storing a ‘monitor’ boolean value in the database record associated with a reference data fingerprint. Users can simply change the boolean value to trigger monitoring drift detection for the corresponding reference data fingerprint.
[0654] User control over the drift-detection is important for users, e.g. a user may have a set of servers (host computers) where the running processes differ, but the list of users (and potentially other user-related attributes such as their SSH keys) is not expected to change. A user running miscellaneous processes is not suspicious, but a new user being created on the system could be.
[0655] The system does not compare data fingerprints that are of a different investigative module

type to that of any of the reference data fingerprints stored. Users may thus configure their drift detection profile to cover a specific scope without generating false positives by ignoring drift in areas that they don't expect to remain the same. This configuration is achieved by the user choosing what data form attributes to include in the reference data fingerprints and/or choosing which reference data fingerprints are to be monitored.

[0656] The reference data fingerprints may be updated as necessary, e.g. when system updates occur or there is a change in policy or for any other reason. The investigation system (**1**) is configured to compare data fingerprints with the most recent corresponding reference data fingerprint, even if the database contains multiple matching reference data fingerprints.

[0657] A particular problem in information security is rapidly responding to a security incident when there is no, or ineffective, security monitoring in place. Often security teams will be presented with thousands of host computers but no way to rapidly determine which host computers should be investigated first, and which host computers can be deprioritized as not impacted. Due to the often-limited size of incident response teams, and even more limited knowledge on investigating Linux systems, the scale of the problem quickly overwhelms all response efforts.

[0658] The present disclosure can ameliorate this problem by using the drift-detection methods as aforementioned.

[0659] The methods can also be applied retrospectively by generating reference data fingerprints for the drift detection from a matching host computer with a known-good baseline state e.g. from an isolated host computer that the user knows is secure, or a freshly installed system using an image the user knows is not compromised.

[0660] As an example, a customer has 1,000 host computers using an Ubuntu Linux 23 distribution. They know that a certain number are likely compromised, but existing security monitoring tools have not shown any problems. The user can use the investigation system (**1**) to generate reference data fingerprints for a known-good Ubuntu Linux 23 host computer in isolation. The investigation system (**1**) can then perform investigations of all 1,000 host computers to generate corresponding data fingerprints and compare with the known-good reference data fingerprints. Any host computers that don't have matching data fingerprints will trigger an alert as there must be some difference in a set of data form attributes, e.g. a new process, user, scheduled task, systemd service, kernel module etc.

[0661] Utilising the method of drift detection and investigation system (**1**) can thereby rapidly reduce the number of systems to investigate.

[0662] Moreover, the differences identified by the investigation system (**1**) indicate what data form attributes are different and thus users can more quickly identify what sort of compromise may be present. A host computer may have a data fingerprint with a data form attribute that was not on the known-good reference data fingerprint. The type and value of the data form attribute provides an indicator of the potential threat, e.g. if the different data form attribute is an executable file with a corresponding process running, then that file may present a threat. Similarly, if the data form attribute was a new SSH key or a new user then these would indicate potential threats or compromises. The ability to quickly reduce the number of attack vectors and systems to a small number amplifies incident response effectiveness greatly.

[0663] Linux is the most common operating system for many embedded systems and appliance applications. The open-source nature of the code and wide CPU support means Linux operating systems can run across multiple industries and use cases such as robotics, cameras, navigation systems, medical devices, network routers, Internet of Thing (IoT) devices, and the like. A security concern for these types of devices is the infrequency that they are updated. This infrequency can be due to general neglect or regulatory requirements that require updates pass extensive and time-consuming re-certification processes.

[0664] In applications where a device is not facing frequent updates, the agentless drift detection methods of the present disclosure are an ideal mechanism to find compromises. As aforementioned,

a reference data fingerprint of a known-good device can be created, e.g. a known-good network router, robot, or IP camera. The reference data fingerprint can then be used as a comparison point with similar devices on the user's network. Any device that shows differences to the reference data fingerprint, such as a new process, new user, new scheduled task, or other change, can immediately be identified as a threat.

[0665] In the above example, the present disclosure's drift detection is effective enough that traditional threat detection signatures commonly found in anti-virus and anti-malware solutions are superseded. The drift detection alone is enough to identify new threats and indeed is often more accurate and reliable as unknown threats operating on embedded systems are immediately spotted for investigation.

[0666] Furthermore, as the investigative modules (**3**) are agentless and are removed on completion, the drift-detection methods may be used on a wide variety of embedded devices, IoT and appliance applications without any risk to stability or performance on critical infrastructure. Complete detection is provided against threats with minimal chance of disruption or deployment issues vs. conventional agent-based protection mechanisms.

[0667] The investigation system (**1**) and drift-detection methods may also be used to detect drift inside Docker and related container technology as is commonly used in Linux systems. Any container running on a host computer can have changes located within the container just as if it were a standalone computer system. The features and benefits of the drift detection apply directly to the containers such as process, file, log, user, systemd, scheduled task, SSH keys and other types of drift that could indicate compromise. Furthermore, drift detected inside a container can be flagged as “containerized”, allowing specialized treatment of compromised container workloads vs. the host operating system running the containers.

[0668] A table of examples of varying levels of drift detection is set out below, showing the type of data form investigated and the default permissive and moderate sets of attributes for each level of drift-detection monitoring.

TABLE-US-00002 Data Permissive Moderate attribute set Form attribute (applied in addition to Type set the permissive attributes) at jobs command username btmp, utmp, wtmp username hostname (system login data) IP address cron jobs command path username directories name owner uid path owner gid mode date created date modified containerized files name owner uid path owner gid “magic number” mode class size actual byte count sha512 hash containerized kernel name module file path modules hidden file owner uid missing file file owner gid taints file mode file sha512 hash lastlog username uid hostname processes name executable sha512 hash path process uid process gid systemd context scope Each of the following for the units context uid start, start_pre, start_post, unit type reload, stop, and stop_post load state exec commands on service units: active state file uid summary of file mode service file sha512 hash exec info associated unit name (if socket unit) systemd username uid sessions linger gid runtime path users username group name uid gid home dir path GECOS field group membership

[0669] These levels can be modified and customised to suit users and their requirements. Additional attributes may be added at each level as required.

[0670] The investigative functions of some exemplary investigative modules (**3**) are included in Appendix A. The investigative modules listed are exemplary only and show functional descriptions of the various investigative modules (**3**). There may be numerous investigative module types that cover a wide range of investigative functions and a major benefit of preferred embodiments of the investigation system (**1**) is that new investigative modules (**3**) may be written and deployed without any modification or software installation on the host computer (**2**).

[0671] More detailed explanations of a selection of investigative modules (**3**) are now described below.

Masquerading Processes.

[0672] Some investigative modules (**3**) may search for illegitimate programs running processes

appearing to be legitimate system processes. Often, such illegitimate processes will use names or other data form attributes identical to system processes to prevent an investigator or administrator from identifying them. Such programs have data form attributes (names or other attributes) differing from a predefined integrity status (identical to system processes) and are thus considered suspicious.

[0673] One method in which the investigative modules (**3**) may locate such illegitimate programs is by generating cryptographic hashes of every running process on the system and analyzing the list to identify identical process names where the cryptographic hashes do not match, i.e. the process condition is suspicious if processes with identical names in the system don't have identical cryptographic hashes. Thus, in this example, the predefined integrity status is complied with if the cryptographic hashes were identical and considered suspicious if the hashes differ.

[0674] For example, if you have two system processes that purport to be a webserver (e.g. “httpd”), then the cryptographic hashes should be identical when operating. However, if we discover that two programs named “httpd” have different cryptographic hashes then we can determine immediately that they are not in fact the same and one is impersonating the other.

[0675] When the process is ascertained as suspicious the investigative module (**3**) will flag both processes for investigation along with comprehensive forensic information in the form of data form attributes such as process names, process dates run, process owner, process locations, process files open, process network sockets open, process cryptographic hashes and similar.

Renamed Processes.

[0676] Some investigative modules (**3**) may search for renamed processes. Sometimes an attacker may modify a data form attribute such as the name of a legitimate program before starting it on the target host computer. This can allow the attacker to use the program for a malicious activity while avoiding detection as the program is renamed and thus appears to be a different legitimate process. For instance, a system network utility that opens a network socket and attaches itself to a system shell as a backdoor is a legitimate program but may be used maliciously. Under this situation the attacker may want this activity to appear to be a legitimate webserver or other system daemon to prevent close inspection.

[0677] In order to locate a renamed process the investigative module (**3**) may build a list of all system processes that are running, along with their hashes. The investigative module (**3**) then retrieves identical system hashes that are also running and ensure they have the same program name. Any hash that is identical to another process, but has a different name, is deemed suspicious, i.e., differing from a predefined integrity status (of having identical hashes).

[0678] For example, an attacker can start the popular and common Secure Shell (sshd) daemon to use as a backdoor onto a host. Most systems have sshd running by default on Transmission Control Protocol (TCP) port 22 so this is not unusual. However, an attacker may start sshd and have it impersonate an Hypertext Transfer Protocol (HTTP) server on port 80. The attacker can then connect to port 80 with their own credentials and get access to the system. However, for an administrator reviewing the system it may appear to be a webserver running on that port.

Masquerading Files.

[0679] ‘Masquerading’ files are suspicious files used in a technique whereby an attacker modifies a data form attribute such as a file name by renaming a legitimate system program to give a misleading appearance on the file system. For instance, they may take a system shell and copy it to a new name and use that as part of a privilege escalation attack.

[0680] Prior art systems have difficulty in identifying if such behavior constitutes suspicious behavior (i.e. differing from a predefined integrity status) without doing an extensive cryptographic hashing of the files on the remote host file system ahead of time. These predefined integrity status checks are slow and extremely CPU intensive. They also require storing cryptographic hashes to be used for comparison later. Further, they are prone to false alarms as systems are updated and new files are patched or introduced. Each time a new set of cryptographic hashes needs to be

regenerated and if this is not done then false alarms or missing malicious activity is possible.
[0681] In contrast, some investigative modules (**3**) in the investigation system (**1**) are written to generate cryptographic hashes of system binaries before hunting for masquerading files.
[0682] In one embodiment an investigative module (**3**) may be configured to retrieve system binary directories such as /bin, /sbin, /usr/bin, /usr/sbin and generate a list of all binaries present and generate cryptographic hashes of each. The investigative module (**3**) may then search on the file system in key critical areas such as temporary and user directories for identical hashes having a different name.
[0683] For example, an attacker can copy the system shell “/bin/bash” to their home directory and rename it “.data” and give it permissions to run as an administrator (SUID root). When the attacker logs into the host, they can simply run the “.data” binary and get instant administrator access. The investigative module (**3**) may find this activity because it first generated hashes of all legitimate binaries and compared it against user home directory files to see if an identical hash was found but with a different name. In the above example, the investigative module (**3**) would ascertain that the file “/bin/bash” matches the malicious user's file “.data” and issue an alert with data relating to the file to assist an investigator.
[0684] The above investigative module (**3**) has advantages over prior art methods in that the hashes are generated on the fly and do not need to be stored on a central server. The investigative module (**3**) may also be used regardless of whether the host computer has been previously investigated. Furthermore, the targeted approach is relatively fast as the investigative module (**3**) performs the investigation for key system areas and doesn't expire because in each execution a fresh list of binary signatures is generated.
SSH Hunter Investigative Module
[0685] SSH keys are associated with specific users and are installed by placing entries in known file locations under the user's home directory. These known locations are the user's SSH ‘authorized_keys’ files.
[0686] In one embodiment, a user investigative module (**3**) is provided that is an ‘SSH hunter’ investigative module that locates data forms provided in the form of SSH Public keys. This SSH hunter investigative module (**3**) process is represented by the flowchart in FIG. **14**. The SSH hunter investigative module (**3**) is run once sent to the host computer i.e., typically performed at stage ‘j’ shown in FIG. **6**.
[0687] As shown in FIG. **14**, the SSH hunter investigative module (**3**) reads the list of users of the host computer (e.g., from the /etc/passwd file) to retrieve the location of users' home directories. The username and user numerical ID are also retrieved with the directory location.
[0688] Where required, the SSH hunter investigative module (**3**) can also be used to retrieve user information from distributed login authentication systems such as Lightweight Directory Access Protocol (LDAP), ote Authentication Dial-In User Service (RADIUS), Active Directory, or similar solutions.
[0689] The SSH hunter investigative module (**3**) sequentially searches the located directories for files matching a key file type. If key files are found, each line of the key file is read to determine if that line corresponds to an SSH key. If an SSH key is found in that line, the SSH key entry is stored in the investigative data.
[0690] Subsequent lines in the key file are read to locate any further SSH keys.
[0691] If no key file is located for the corresponding user account or if no further lines are detected in the key file, then the loop is exited.
[0692] The SSH hunter investigative module (**3**) then determines if more user accounts were detected and if so, repeats the steps of searching the directories of authorized key files for each user account located.
[0693] Once all key files have been located and read, the SSH hunter investigative module (**3**) returns a JSON file of the investigative data, and the investigation may be resumed using any other

investigative modules (**3**) pushed to the host computer (**2**).
[0694] The investigative data results (**4**) regarding the SSH key files are passed to the scanning node (**6**) for storage, reporting and analysis.
[0695] The SSH keys retrieved in the investigative data are recorded by the server (**5**) in database (**7**). This process is represented in FIG. **15**, section A.
[0696] The retrieved keys are each stored into the database in multiple tables. These tables include an 'SSH_PUBLIC_KEY' table, a 'USER_KEYS' table, and a 'USER_KEY_ENTRIES' table. These table names are arbitrary and exemplary only.
[0697] The SSH_PUBLIC_KEY table stores the key without relations to a corresponding host or user.
[0698] The USER_KEYS table is a pivot table with relations to the key in the SSH_PUBLIC_KEY table along with relations to a corresponding user account and the host the key was retrieved from.
[0699] The USER_KEY_ENTRIES table stores the key and specific line of the key file along with a relation to the host the key was retrieved from.
[0700] These tables are explained in more detail below.
[0701] The SSH_PUBLIC_KEY table stores SSH public keys as independent records, i.e. without relations to the corresponding host they appear on, or users who are using the keys. Keys are identified by the sha512 hash of the key value itself, regardless of key options, comments or other attributes. This table may be searched to locate duplicate key entries, indicating the same key is used in multiple places.
[0702] As shown in FIG. **15**, when adding a key to a new record in the SSH_PUBLIC_KEY table, the server (**5**) searches the SSH_PUBLIC_KEY table for a matching key and if not present, adds the record along with the current time, recorded as a timestamp indicating that the record is the 'first seen' or first located incidence of the key.
[0703] If an existing matching record is present, then the existing record is updated with the current time recorded as a 'last seen' timestamp.
[0704] The SSH_PUBLIC_KEY table thus provides usage information regarding the keys stored, e.g., if the key has even been used by any user on any host, as well as the extent of the time range the key appeared within.
[0705] SSH keys have no inherent recognizable metadata or value other than the algorithm and "big number" that is the value of the particular key. The server (**5**) is therefore configured to generate and record a "friendly key name" in the database (**7**) for each key. The "friendly key name" is based on the first nine bytes of the sha512 hash of the key value. The friendly name is generated by randomly retrieving three words from numerical positions x, y, and z in a 'dictionary' file. x is the integer value of the first 12 bits of the sha512 hash, y is the integer value of the second 12 bits of the sha512 hash, and z is the integer value of the third 12 bits of the sha512 hash. Since the sha512 hash of a key is effectively a uniformly distributed random number, the first 36 bits provide for over 68 billion possible friendly names. It is unlikely two SSH keys in actual use will generate the same friendly name.
[0706] The friendly name allows an operator, reading the table to more easily recognize a key. For example, a user is far more likely to recognize "tapping frugality naturist" a second time during an SSH Hunter session than the underlying key hash that corresponds to that name, which starts with dba5a9869c076e5 . . . .
[0707] The USER_KEY table stores keys in records with relations to the corresponding user and host. The USER_KEY table thus records where keys appear at a per-user-per-host level.
[0708] Each key retrieved is recorded in the USER_KEY table with foreign id fields corresponding to the corresponding user and host that the key was retrieved from. As per the SSH_PUBLIC_KEY table, the key is stored in a record with timestamps corresponding to 'first seen' if no matching record is found and 'last seen' if a record is located with the same key, user and host.
[0709] After each investigation of a host, if the USER_KEY table contains keys with no matches

from the investigation, the record is updated to mark those keys as being ‘removed’. A ‘removed’ key indicates that the key is no longer found in the key file. This removal could indicate the user has stopped using the key. The key record is retained in the USER_KEY table so that operators may retrieve the information and determine that the removed key was previously active at some point in the past i.e., between the ‘first seen’ and ‘last seen’ timestamps.
[0710] A new USER_KEY record is created if a removed key appears in a further investigation for the same user on the same host. This new key is not marked as removed and resides alongside the ‘removed’ key. Keeping records of when keys are removed and reappear is useful in providing a timeline of when the key permitted access to the corresponding user on the corresponding host.
[0711] The USER_KEY table records are related to the HOST entity by the unique host ID that the investigation system (**1**) generates for each host investigated. The relation allows related host information to be queried, e.g. What types of hosts can a user access? what Linux and distribution versions is the host running? does the host have any security alerts or errors?
[0712] The data in the USER_KEY table enables database queries to be performed easily to quickly answer questions such as “what hosts does a particular username have access to?”, “what usernames have access to a particular host id?”, and “what hosts have users that a particular SSH key grants access to?”
[0713] The USER_KEY_ENTRY table records details about the specific lines in key files that allow the access described by entries in the USER_KEY table. Every user that has access to a host by means of an SSH key is due to one or more specific entries in a key file. If an operator identifies unneeded or malicious SSH keys, by means of exploring the data derived from the USER_KEY table, the data in the USER_KEY_ENTRY table will indicate to the operator where they need to remove the keys from in order to revoke the unauthorized or unneeded access.
[0714] Nothing in the SSH protocol prevents users from adding duplicate keys to the same file. If there are multiple copies of the same key, it may mislead operators who are trying to revoke access, i.e., they may remove the first copy of the key they see in the authorized keys file, falsely thinking they have therefore revoked access, while additional copies of the key may be stored elsewhere and still allow the private key holder access to the host (**2**).
[0715] The data in the USER_KEY_ENTRY table thus enables operators to identify all the key file entries that contain a particular key instead of having to manually parse every key file.
[0716] The SSH hunter investigative module (**3**) and database (**7**) may thus provide information necessary to identify and monitor SSH keys in host computers (**2**) and thereby identify potential compromises or malicious activity using SSH keys. Some examples of the queries that can be performed on data retrieved with the SSH hunter investigative module (**3**) are included below.
[0717] If malware is known to leave specific SSH keys on systems to gain access in the future, a query for that specific SSH key may be performed and all copies of that SSH key found on host computers (**2**) can be easily identified and removed if necessary.
[0718] If operators have a policy that credentials shouldn't be shared between users, all SSH keys that are shared by more than one user can be easily identified from querying the USER_KEY table for records matching the same key but different users.
[0719] If a host (**2**) is suspected to be compromised, a query of the USER_KEY table can locate SSH keys that allow access to that host (**2**) and the corresponding users that have access to that key identified. Any one or more of those users may be a potentially malicious user. The query can be extended to identify any other hosts that the potentially malicious users have SSH key access to and thereby provide information to understand the potential impact of the compromise beyond the initially identified host, or whether suspicious SSH keys may be involved in the compromise or attack.
[0720] A query of the USER_KEY table can be performed for users that have a threshold number of keys stored against their id. This threshold can be used to determine if users have an unusual number of keys, potentially indicating the user may use automated methods to create and install

SSH keys frequently without removing old keys.
[0721] A query of the USER_KEY table can be performed for users or hosts if those hosts or users are determined to be subject to policies that require newer encryption/public key algorithms be used for SSH keys (e.g., ed25519 instead of Rivest-Shamir-Adleman (RSA)). The retrieved results can provide a list of all SSH keys by user or host that are out of compliance with that policy.
[0722] Operators would normally manually disable or remove a user's account on hosts if that user is determined to no longer have access to an SSH key (e.g., their employment is terminated). The SSH hunter investigative module (**3**) enables a query to be performed of the USER_KEY table for the user to determine all the SSH keys associated with the user, some of which may not have been known to the operator.
[0723] The of the USER_KEY table or SSH_PUBLIC_KEY tables can be queried for SSH keys that are known to be malicious and retrieve the locations from the USER_KEY_ENTRY table so that the keys can be removed.
[0724] Details of SSH keys that are determined to be as malicious can be published to indicate to other operators that those SSH keys are hostile/compromised.
[0725] A BANNED_SSH_KEY table may be used that records SSH keys that are banned for a particular host or user and then the USER_KEY_ENTRY table queried after the SSH hunter investigative module (**3**) is run to determine if any of the retrieved SSH keys match records in the BANNED_SSH_KEY table, thereby indicating a banned key is being used.
[0726] Queries can be run for retrieving records where timestamps are within a particular time range, thus enabling information to be retrieved indicating keys that have appeared within that time range. This query can be refined to filter for specific user accounts that are normally restricted (e.g. root or admin users).
[0727] FIG. **16** shows a page of a user interface displaying data about an SSH Key. The user interface displays useful information regarding the SSH key, including: [0728] the number of ‘active’ user accounts having the key, i.e., those user accounts that currently have the key in their authorized keys file; [0729] the number of ‘All-Time’ user accounts that had the key, i.e., this includes all accounts that have ever had the key, not just those with the key currently; [0730] the number of ‘active’ host computers having the key, i.e., those host computers that currently have the key; [0731] the number of ‘All-Time’ host computers that had the key, i.e., this includes all host computers that have ever had the key, not just those with the key currently; [0732] when the key was last retrieved in investigative data; [0733] a timeline of the key usage; [0734] host computer details of host computers that have the key.
[0735] FIG. **17** is another user interface page showing the data stored in the database from the key, user and host perspectives.
[0736] The key investigation table shows records of the keys, corresponding friendly name, key type, number of active users, number of active hosts, number of active hosts which have alerts as well as temporal data in the form of first seen and last seen timestamps.
[0737] The user investigation table shows records of the user accounts, username, number of active keys, total number of keys found over all time, number of ‘active’ host computers having a public key related to the user account, total number of host computers having a public key related to the user account and temporal data in the form of first seen and last seen timestamps for keys found related to that user account.
[0738] The host investigation table similarly shows records of the host computers, host name, address, no of active user accounts related to keys on the host computer, total no of user accounts found over all time, no of active keys, total number of keys found over all time and the operating system type.
[0739] total number of host computers having a public key related to the user account and temporal data in the form of first seen and last seen timestamps for keys found related to that user account;
[0740] In contrast to manual monitoring of SSH keys, the SSH hunter investigative module (**3**) and

associated database tables enable rapid threat response as SSH key threats can more quickly be located across many hosts and users. Moreover, the information contained in the tables can be used to generate reports about key usage, types of keys, new keys, old keys to help management teams identify and track SSH keys.

Cloaked Stealth Rootkit

[0741] Stealth rootkits on the Linux platform generally operate as what we call Loadable Kernel Modules (LKM). An LKM rootkit is able to intercept system calls to hide its presence and activity. For instance, the rootkit can intercept system file read operations so when an attempt is made to look into key system configuration files the rootkit can hide flagged lines from operation. In one method, an LKM rootkit may hide its start-up mechanism in this way so investigators cannot see how the rootkit initializes itself when the system starts. Once a system boots, the LKM rootkit ensures investigators are unable to see this information to remove it and prevent it from restarting.

[0742] LKM rootkits typically intercept system calls using standard file read/write operations or directory read/write operations.

[0743] An investigative module (**3**) may be included in the investigation system (**1**) to combat such rootkit cloaking techniques by bypassing standard file operations and instead uses direct memory mapped I/O. Memory mapped file I/O is more difficult for a malicious LKM rootkit to intercept and can be used to avoid cloaking techniques used by such rootkits. Moreover, memory mapped file I/O is much faster than standard file I/O operation. Thus, such an investigative module (**3**) can be used by other investigative modules for performance reasons as well as preventing future stealth rootkit activity hiding from detection.

Network Sniffers and Packet Capture (PCAP) Files

[0744] Hidden network sniffers are used to extract network traffic for malicious reasons such as stealing credentials. Many of these sniffers use built in libraries to accomplish their task with the primary library used being called libpcap that generates a file format called “PCAP” (short for Packet CAPture).

[0745] An investigative module (**3**) as used herein may identify such network sniffers by searching the process table for programs with open files, then analyzing the open files to see if they are of the PCAP file format. If a program has a file open that is of the PCAP file type, then it is flagged as a possible sniffer operating on the host computer (**2**).

[0746] PCAP files also represent remnants of the network sniffers as they are used to store captured network data. While there are legitimate reasons for these files to be present on a host (e.g. network debugging), there are also a lot of malicious reasons which makes their presence suspicious.

[0747] An investigative module (**3**) as used herein may be configured to search specifically for hidden PCAP files or PCAP files in known suspicious locations such as /tmp, /dev/shm, and other critical system areas commonly targeted by Linux malware and rootkits.

Tampered or Invalid Login Shells

[0748] Another method used by attackers to maintain access to a host is to replace disabled accounts with valid login shells. For instance, on Linux, users with the shell named “/sbin/nologin” or “/bin/false” are presumed to not allow system login. However, an attacker can replace these binaries with a valid shell like “/bin/bash” and it could allow them to login even though to an administrator it appears as if they cannot.

[0749] An investigative module (**3**) may thus be configured to generate cryptographic hashes of all valid system shells such as /bin/bash, /bin/sh, /bin/csh, /bin/zsh, etc. The investigative module (**3**) may then check the common disabled login commands such as “nologin” and “false” to be sure they do not match the hashes of valid shells. If a match is found, then it is presumed the shell was renamed to an invalid type and an alert is generated with relevant forensic data.

Missing Login Audit Log Entries

[0750] Many attackers will attempt to remove themselves from system login audit logs after they gain entry to the host. This prevents administrators from seeing suspicious logins and helps the

attacker conceal their activity.
[0751] The most common way to delete a login entry is to remove it from the system “wtmp” and “utmp” files. These two files keep login data for active and previous logins into the system.
[0752] There is a third login auditing mechanism however called “lastlog.” Lastlog keeps only the last login of a user whereas logs like wtmp keeps a record of all logins in time sequence for all users. While lastlog is useful for finding when a user last logged in, it is not useful to see a list of all logins and the times they did it.
[0753] Many log cleaning tools focus on removing entries from wtmp/utmp but leave lastlog untouched. As a result, investigative modules (**3**) can be created to detect such tampering.
[0754] An investigative module (**3**) may be configured to create a table of all users in the lastlog file. This table will include the last time users logged in. The investigative module (**3**) then creates a table of last logins from the wtmp file. They two lists are compared. If a user entry was found in lastlog, but the entry is missing from wtmp for the same time period, then the investigative module (**3**) determine that the user was deleted from the wtmp log and indicates the log is suspicious.
[0755] For example, a user called “john” logs in at 12:05 AM. This is recorded in wtmp and lastlog. The user john then deletes their login from wtmp at 12:05 AM but keeps previous logins untouched to not arise suspicion. However, they do not remove the login from lastlog.
[0756] The investigative module (**3**) may build a table from lastlog showing user “john” logged in at 12:05 AM on that day. Then it will build a table of logins for “john” from wtmp and show that the last login saved was at, for example, 9:17 PM the day before. Because the 12:05 AM entry is missing in wtmp, but not lastlog, then an alert is generated that the entry “john” was deleted maliciously.

Suspicious Log Cleaning

[0757] Many pieces of malware make an incomplete attempt to clean their presence by deleting all system log entries and replacing them with files that are zero bytes in size. This can result in a dozen or more files removed and it is very hard to recover the data to discover what happened.
[0758] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to ascertain such activity by looking at common system log areas such as /var/log on the target system. This investigative module (**3**) will count the number of audit logs that exist and compare it against the file sizes. If the investigative module (**3**) determined a preponderance (e.g. >50%) of zero-byte sized logs then the investigative module (**3**) returns investigation data indicating the logs are suspicious. It is very unusual for Linux to have many log files that are all concurrently zero bytes long which makes this kind of investigative module particularly useful.

Bindshell Backdoors

[0759] ‘Bindshell backdoors’ are a way for attackers to gain access to a remote host. They are methods that use built-in programs on the targeted system to connect back to an attacker's machine or open a local network port to allow the attacker to connect and then gain access.
[0760] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to retrieve and analyse the process table and analyze common tools to see if they are operating as bindshells. In particular, the investigative module (**3**) may search common tools such as netcat, socat, telnet, python, ruby, perl, and similar utilities to see if they are running a system shell, are connected to a network port, and have open file descriptors commonly used for bindshell activity. If the investigative module (**3**) determines such combinations of data form attributes the investigative module (**3**) returns investigation data indicating the process is suspicious.

Detecting Deleted Processes

[0761] Linux malware in the form of a binary will often install itself onto a system and then delete the binary from the disk to conceal its presence. Once memory-resident, the binary does not need to be on the disk. By deleting itself, the binary avoids detection by common file scanning and file integrity methods.
[0762] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to

search for deleted processes running on the remote host. While there are some legitimate reasons why a binary may be deleted (such as during an upgrade), deletions are mostly for malicious reasons. The investigative module (**3**) searching for such activity will go through the process table to get the full path to the running binary. The investigative module (**3**) will go directly to the binary path to ensure it exists on the system to avoid false alarms. If the investigative module (**3**) does not locate the binary on the disk, but it is still present in memory, then the investigative module (**3**) returns investigation data indicating the corresponding process is suspicious.

Detecting Processes with Listening Network Ports in Suspicious Locations

[0763] Many pieces of malware will setup network operations once active. At the same time, these binaries will be located in system areas that are unusual and therefore suspicious. For instance, the malware may be located in system/tmp or /dev directories.

[0764] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to search for this kind of activity. If the investigative module (**3**) detects a process running from a suspicious location with network activity such as /tmp, /dev, or other critical system areas not normally used for this purpose, then the investigative module (**3**) returns investigation data indicating the corresponding process is suspicious.

Running Processes from Suspicious Directories or Names

[0765] Many pieces of malware will try to hide their presence by loading themselves into hidden or obscure system directories. This is an attempt to prevent discovery of the binary and associated data by making them difficult to locate on the file system.

[0766] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to search for any process running from a location in a list of locations considered as unusual or suspicious locations. For instance, the investigative module (**3**) may identify processes running from hidden directories, from unusual system directories commonly used by malware (such as /tmp or /dev/shm), or processes named in unusual ways. In the last example, the investigative module (**3**) may identify suspicious processes as processes named as a hidden file, with only one character as the name, named with spaces, named as another process but with a different case (e.g. crond vs. Crond), and similar permutations. The investigative module (**3**) may then return investigation data indicating the suspicious process.

Python, Perl, PHP: Hypertext Preprocessor (PHP), and Ruby Scripts Running in Unusual Locations

[0767] Many malware types after a website compromise will inject malicious code to be run by the server in a scripting language like Python, Perl, PHP, or Ruby. Often, this code will run out of the system/tmp or /dev/shm directories because they are world writable by the webserver.

[0768] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to search for Python, Perl, PHP, or Ruby scripts running out of /tmp or /dev directories as this is considered extremely suspicious behavior on Linux systems. The investigative module (**3**) may search for Python, Perl, PHP or Ruby code but any scripting language or binary can be used in a similar way. When the investigative module (**3**) detects these types of programs running from/tmp or /dev directories the investigative module (**3**) returns investigation data indicating the corresponding process is suspicious.

Strace Running

[0769] Strace is a tool used to trace a running program for debugging purposes. However, it can be re-purposed to extract confidential data from a running program by having the tool attach when the program runs. For example, strace can be attached to the system “ssh” program to extract passwords when the user calls ssh to login to another host.

[0770] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to search for any process with strace attached to it for more than a predetermined period of time—e.g. two minutes. The time delay is used so that the investigative module (**3**) doesn't generate an alert for legitimate strace users when debugging a process, which normally takes a short time, while the investigative module (**3**) does generate investigation data indicating the corresponding process is

suspicious if a malicious user keeps strace attached to extract large quantities of data.

Detecting Anti-Forensics in Use with a Process

[0771] Anti-forensics are methods attackers use to prevent analysis and detection of their activity. On Linux one common anti-forensics tactic is to prevent logging command history by setting the history file environment variables such as HISTFILE to not log data by dumping the variable to places like /dev/null or setting it to be zero bytes allowed. These environment variables are attached to every process the user subsequently executes and the commands they used during that time are not saved.

[0772] An investigative module (**3**) may thus be configured to investigate the host computer (**2**) to search for processes that have environment variables set that indicate anti-forensics are in use. For example, the environment variable HISTFILE=/dev/null is a typical anti-forensic method. The investigative module (**3**) may be configured to analyse all running processes and if the investigative module (**3**) determines an environment variable indicates anti-forensics are in operation, the investigative module (**3**) returns investigation data indicating the corresponding process is suspicious.

[0773] It should be understood that there exist implementations of other variations and modifications of the disclosure and its various aspects, as may be readily apparent to those of ordinary skill in the art, and that the disclosure is not limited by specific embodiments described herein. Features and embodiments described above may be combined with and without each other. It is therefore contemplated to cover any and all modifications, variations, combinations or equivalents that fall within the scope of the basic underlying principals disclosed and claimed herein.

APPENDIX A

Investigative Module Functions

[0774] core dependency file for investigative modules to use. [0775] boot dependency file for investigative modules to use. [0776] looks for hidden directories in /bin, /sbin, /usr/bin, and /usr/sbin. [0777] looks for hidden directories in the /dev directory [0778] looks for hidden directories in the /dev/shm directory [0779] looks for hidden directories in /lib, /var/lib, etc. [0780] looks for hidden directory names that are extremely suspicious anywhere on the file system. [0781] looks for hidden directory names that are extremely suspicious under /bin directories. [0782] looks for hidden directory names that are extremely suspicious under /dev directories. [0783] looks for hidden directory names that are extremely suspicious under /etc directories. [0784] looks for hidden directory names that are extremely suspicious under /lib directories. [0785] looks for hidden directory names that are extremely suspicious under the /top-level directory. [0786] looks for hidden directory names that are extremely suspicious under /run and /var/run directories. [0787] looks for hidden directory names that are extremely suspicious under /boot, /sys, and /lost+found directories. [0788] looks for hidden directory names that are extremely suspicious under /tmp directories. [0789] looks for hidden directory names that are extremely suspicious under a user's home directory. [0790] looks for hidden directory names that are extremely suspicious under /usr/games and /usr/share/games directories. [0791] looks for hidden directory names that are extremely suspicious under /usr/include [0792] looks for hidden directory names that are extremely suspicious under /usr/local directories. [0793] looks for hidden directory names that are extremely suspicious under /usr/share directories. [0794] looks for hidden directory names that are extremely suspicious under /var directories. [0795] looks for hidden directories in various system directories (/boot, /lost+found) [0796] looks for hidden directories under /usr/games or /usr/share/games [0797] looks for hidden directories under /usr/share/man [0798] looks for an inconsistent link count anywhere on the file system. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0799] looks for an inconsistent link count for bin directories. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0800] looks for an inconsistent link count for dev directories. This means there is a directory

present, but it is being hidden from view likely by a stealth rootkit. [0801] looks for an inconsistent link count for/etc. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0802] looks for an inconsistent link count for top level system lib directories. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0803] looks for an inconsistent link count for the top level/directory. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0804] looks for an inconsistent link count for top level system directories. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0805] looks for an inconsistent link count for the top level/usr directory. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0806] looks for an inconsistent link count for/usr/games, /usr/share/games. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0807] looks for an inconsistent link count for/usr/local. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0808] looks for an inconsistent link count for/usr/share. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0809] looks for an inconsistent link count for the top level/var directory. This means there is a directory present, but it is being hidden from view likely by a stealth rootkit. [0810] looks to see if a system shell has been renamed to/bin/false to hide the fact that an account can login. [0811] looks if there is an encrypted or packed binary anywhere on the file system. [0812] looks if there is an encrypted or packed binary in system cron directories. [0813] looks if there is an encrypted or packed binary in system dev directories. [0814] looks if there is an encrypted or packed binary in system etc directories. [0815] looks if there is an encrypted or packed binary in system run directories. [0816] looks if there is an encrypted or packed binary in system/boot, /lost+found, and similar directories. [0817] looks if there is an encrypted or packed binary in system temp directories. [0818] looks for executable files hidden anywhere on the file system. [0819] looks for executable files hidden in system binary directories. [0820] looks for executable files hidden in system/dev directory. [0821] looks for executable files hidden in the /etc directory. [0822] looks for executable files hidden in system lib directories. [0823] looks for executable files hidden in the top level directory. [0824] looks for executable files hidden in system/run directory. [0825] looks for executable files hidden in system temp directories. [0826] looks for executable files in system cron directories. [0827] looks for executable files in system/dev directory. [0828] looks for executable files in system/etc directory. [0829] looks for executable files in system run directories. [0830] looks for executable files in system temp directories. [0831] looks for Linux executables named as a common non-executable extension to masquerade their presence anywhere on the file system. [0832] looks for Linux executables named as a common non-executable extension to masquerade their presence in a system device directory. [0833] looks for Linux executables named as a common non-executable extension to masquerade their presence in the /etc directory. [0834] looks for Linux executables named as a common non-executable extension to masquerade their presence in a system library directory. [0835] looks for Linux executables named as a common non-executable extension to masquerade their presence in the top level root directory. [0836] looks for Linux executables named as a common non-executable extension to masquerade their presence in the /run directory. [0837] looks for Linux executables named as a common non-executable extension to masquerade their presence in the /boot or /lost+found directories. [0838] looks for Linux executables named as a common non-executable extension to masquerade their presence in a system temp directory. [0839] looks to see if a system binary is in /dev where it shouldn't be. [0840] looks to see if a system binary is in /etc where it shouldn't be. [0841] looks to see if a system binary is in the top level directory where it shouldn't be. [0842] looks to see if a system binary is in /run or /var/run where it shouldn't be. [0843] looks to see if a system binary is in a system dir like /boot, /lost+found, and similar where it shouldn't be. [0844] looks to see if a system binary is in /tmp where it shouldn't be. [0845] looks to see if a system binary is in /usr/games or /usr/share/games where it shouldn't be. [0846] looks to

see if a system binary is in /usr/share/man where it shouldn't be. [0847] looks to see if a system binary is in /var where it shouldn't be. [0848] looks to see if a system binary is linked from dev directories. [0849] looks to see if a system binary is linked from temp directories. [0850] looks for system commands that have been poisoned to run malicious code when executed. [0851] looks to see if a system binary has been renamed to a hidden file that still resides in system bin directories. [0852] looks for common files masquerading as one type when they are really another type anywhere on the file system. [0853] looks for loadable kernel module config files that are being altered by a stealth rootkit to hide entries. [0854] looks for packet capture pcap files anywhere on the file system. [0855] looks for hidden packet capture pcap files anywhere on the file system. [0856] looks for packet capture pcap files in the system binary directories. [0857] looks for packet capture pcap files in the system cron directories. [0858] looks for packet capture pcap files in the /dev directory. [0859] looks for packet capture pcap files in the system/etc directory. [0860] looks for packet capture pcap files in the system library directories. [0861] looks for packet capture pcap files in the top-level system root directory. [0862] looks for packet capture pcap files in the system run directories. [0863] looks for packet capture pcap files in the system temp directories. [0864] looks for packet capture pcap files in the system/var directories. [0865] looks for a variety of common Linux rootkit files and directories present on a system. [0866] looks to see if a system shell has been renamed to/sbin/nologin or /usr/sbin/nologin to hide the fact that an account can login. [0867] looks to see if a system shell has been renamed to something else and put anywhere on the file system. [0868] looks to see if a system shell has been renamed to something else and put in the system binary directories. [0869] looks for common start-up scripts that have cloaked entries from a stealth rootkit. [0870] looks to see if a SUID root binary is present in /dev directories. [0871] looks to see if a SUID root binary is present in /etc directories. [0872] looks to see if a SUID root binary is present in /run directories. [0873] looks to see if a SUID root binary is present in /boot, /sys and /lost+found. [0874] looks to see if a SUID root binary is present in /tmp directories. [0875] looks to see if a SUID root binary is present in /usr/games or /usr/share/games directories. [0876] looks to see if a SUID root binary is present in /usr/share directories. [0877] looks to see if a SUID root binary is present in /usr/share/man directories. [0878] looks for all SUID or SGID for any user binaries on the disk [0879] looks to see if a common system editor like vi or nano is set SUID or SGID for any user to enable privilege escalation. [0880] looks for all SUID or SGID root binaries on the disk [0881] looks to see if a common system shells have been SUID or SGID to any user. [0882] looks for suspicious named pipe device files under system binary directories. This is common with some kinds of backdoors. [0883] looks for suspicious named pipe device files under /dev directories. This is common with some kinds of backdoors. [0884] looks for suspicious named pipe device files under /etc directories. This is common with some kinds of backdoors. [0885] looks for suspicious named pipe device files system library directories. This is common with some kinds of backdoors. [0886] looks for suspicious named pipe device files under /run directories. This is common with some kinds of backdoors. [0887] looks for suspicious named pipe device in system directories /boot, /sys and /lost+found. This is common with some kinds of backdoors. [0888] looks for suspicious named pipe device files in /tmp. This is common with some kinds of backdoors. [0889] looks for process PID files that are really binary executable files in disguise. [0890] looks for process PID files that appear to be encrypted data and not process information. [0891] looks for process PID files that contain more than just a standard integer value. [0892] looks for process PID files that are too big to contain only running process data. [0893] looks for evidence that user entries were zeroed out from the btmp file to hide login activity. [0894] looks for log cleaning dropper files left behind in /tmp. [0895] Compares lastlog entries against wtmp entries to see if any have been removed to conceal login activity. [0896] looks for signs the MIG logcleaning tool has been run on the host. [0897] looks for sloppy log tampering such as deleting system logs and replacing with files 0 bytes long. [0898] looks for evidence that user entries were zeroed out from the utmp file to hide login activity. [0899] looks to see if the

system wtmp and lastlog audit records have been erased and made zero bytes long [0900] looks to see if the system wtmp, utmp, and lastlog files are missing. Deleting these files disables login accounting on the system to hide activity. [0901] looks for evidence that user entries were zeroed out from the wtmp file to hide login activity. [0902] returns remote OS version information. [0903] looks for system shells operating as a reverse or standard bindshell backdoor. [0904] looks for netcat running as a reverse or standard bindshell backdoor on the system. [0905] looks for perl scripts running as a reverse or standard bindshell backdoor on the system. [0906] looks for php scripts running as a reverse or standard bindshell backdoor on the system. [0907] looks for python scripts running as a reverse or standard bindshell backdoor on the system. [0908] looks for ruby scripts running as a reverse or standard bindshell backdoor on the system. [0909] looks for telnet running as a reverse or standard bindshell backdoor on the system. [0910] looks for any process that is running with a listening network port, but has been deleted from the disk. [0911] looks for any process that is running with a raw socket, but has been deleted from the disk. [0912] looks for process that is running with a connected outbound port, but has been deleted from the disk. [0913] looks for processes that are running, but the executable has been deleted from the disk. [0914] checks any running process for signs that history file anti-forensics are in use. [0915] looks for processes listening on a network port running out of dev directories. [0916] looks for processes listening on a network port running out of the /proc directory. [0917] looks for processes listening on a network port running out of tmp directories. [0918] looks for any process that is running with a raw socket listening. This could be a backdoor or other malicious program. [0919] looks for any process that is running with raw sockets listening for ICMP packets. This could be a sniffer or backdoor. [0920] looks for any process that is running with raw sockets listening for TCP packets. This could be a sniffer or backdoor. [0921] looks for any process that is running with raw sockets listening for UDP packets. This could be a sniffer or backdoor. [0922] looks for any process that is running with raw sockets listening for unknown protocols. This could be a sniffer or backdoor. [0923] looks for any process that is identical to another running process but has a different name. [0924] looks for any process that is using mixed case in the name to masquerade as another process. [0925] looks for a process that is really netcat, but is masquerading under a different name. [0926] looks for a process that is really a shell, but is masquerading under a different name. [0927] looks for a process that is really socat, but is masquerading under a different name. [0928] looks for a process that is really strace, but is masquerading under a different name. [0929] looks for a process that is really tcpdump, but is masquerading under a different name. [0930] looks for loadable kernel modules that are hiding from view by a stealth rootkit. [0931] looks for processes running with a pcap packet capture file open on the disk operating as a sniffer. [0932] looks for scheduled at jobs that are suspicious or malicious. [0933] looks for scheduled cron tasks that are suspicious or malicious. [0934] looks for processes that are named as a Unix hidden file that are running (e.g. period as start of name) [0935] looks for processes that are running out of /dev. [0936] looks for processes that are running out of a hidden directory under a system binary directory. [0937] looks for processes that are running out of a hidden directory under the /dev directory. [0938] looks for processes that are running out of a hidden directory anywhere in their path. [0939] looks for processes that are running out of a hidden directory under the /etc directory. [0940] looks for processes that are running out of a hidden directory under a system library directory. [0941] looks for processes that are running out of a hidden directory under the root (/) level directory. [0942] looks for processes that are running out of a hidden directory under the /run directory. [0943] looks for processes that are running out of a hidden directory under a system directory such as /boot, /lost+found or /sys. [0944] looks for processes that are running out of a hidden directory under a system temp directory. [0945] looks for processes that are running out of a hidden directory under the /usr directory. [0946] looks for processes that are running out of a hidden directory under the /var directory. [0947] looks for processes that are running out of /proc. [0948] looks for processes that are running out the root user's home directory. [0949] looks for processes that are

running from a suspiciously named directory. [0950] looks for processes that are running out of /boot, /sys and /lost+found directories. [0951] looks for processes that are running out of /tmp directories. [0952] looks for processes that are named as just one character which is commonly done with malware. [0953] looks for perl scripts running out of dev directories. [0954] looks for perl scripts running out of tmp directories. [0955] looks for php scripts running out of dev directories. [0956] looks for php scripts running out of tmp directories. [0957] looks for python scripts running out of dev directories. [0958] looks for python scripts running out of tmp directories. [0959] checks running shell processes for signs that history file anti-forensics are in use. [0960] looks for the strace command running on the remote host for an extended period. [0961] looks for the strace command being used as a keylogger against SSH. [0962] looks for processes with a whitespace as a name. This can conceal a malicious process name as one that looks legitimate. [0963] looks for processes with a whitespace at the beginning or end of a name. This can conceal a malicious process name as one that looks legitimate. [0964] looks to see if the user's history is linked to/dev/null which will conceal command history. [0965] looks for inactive accounts with a valid shell history file in their home directories indicating a login has happened. [0966] checks if a user's home directory is/dev/null which sometimes is done by attackers to conceal activity. [0967] lists root UID 0 users that are not named root which is done to hide superuser accounts. [0968] looks for any user with no password set. [0969] checks user login and logout scripts for anti-forensic tactics to prevent logging their command history. [0970] checks user login scripts for malicious commands that can compromise the system. [0971] looks for inactive accounts with valid ssh login keys in their home directory. [0972] is a dependency for investigative modules to use to assess system files. [0973] is a dependency file for investigative modules to use to look at system process information.

Claims

1. A method of investigating a target host computer, the method using an investigation system comprising a computer system with a computer processor coupled to a system memory and programmed with computer readable instructions, the investigation system being remote to the target host computer, the method using at least one investigative module, wherein the at least one investigative module includes an agentless computer program comprising computer readable instructions to: perform at least one investigative function on a host computer, the at least one investigative function relating to at least one data form, return investigation data relating to the at least one investigative function, the investigation data including at least one data form attribute relating to the at least one data form, and wherein the method includes: performing a reference computer investigation of a reference host computer, the computer investigation including: (i) the investigation system establishing a connection between the investigation system and the reference host computer, and (ii) the investigation system sending the at least one investigative module to the reference host computer, the at least one investigative module performing the at least one investigative function and returning reference investigation data; performing a target computer investigation of the target host computer, the target computer investigation including: (i) the investigation system establishing a connection between the investigation system and the target host computer, and (ii) the investigation system sending the at least one investigative module to the target host computer, the at least one investigative module performing the at least one investigative function and returning comparison investigation data; and comparing a reference data fingerprint to a comparison data fingerprint to determine if said reference and comparison data fingerprints are identical; and wherein said reference data fingerprint includes at least two data form attributes of the reference investigation data; and wherein said comparison data fingerprint includes at least two corresponding data form attributes of the comparison investigation data.
2. The method of claim 1, wherein the reference host computer is the target host computer.

3. The method of claim 1, wherein: (i) said reference data fingerprint comprises a hash of the at least two data form attributes of the reference investigation data, and (ii) said comparison data fingerprint comprises a hash of the at least two corresponding data form attributes of the comparison investigation data.
4. The method of claim 1, wherein at least one of: (i) said reference data fingerprint is comprised of a combination or concatenation of the at least two data form attributes of the reference investigation data, and (ii) said comparison data fingerprint is comprised of a combination or concatenation of the at least two data form attributes of the comparison investigation data.
5. The method of claim 4, wherein: (i) said reference data fingerprint comprises a hash of the combination or concatenation of the at least two data form attributes of the reference investigation data, and (ii) said comparison data fingerprint comprises a hash of the combination or concatenation of the at least two corresponding data form attributes of the comparison investigation data.
6. The method of claim 1, wherein at least one of said data fingerprints is comprised of one of the following: (i) the corresponding at least two data form attributes; (ii) combination or concatenation of the corresponding at least two data form attributes; (iii) hash of the corresponding at least two data form attributes; and (iv) hash of the combination or concatenation of the corresponding at least two data form attributes.
7. The method of claim 1, wherein multiple reference computer investigations are performed to create multiple reference data fingerprints, each of said multiple reference data fingerprints representing a different set or collation of multiple data form attributes.
8. The method of claim 1, wherein the at least one investigative module includes a first investigate module and a second investigative module, the second investigative module including data readable by the first investigative module, the data including instructions or data necessary for the first investigative module to perform the at least one investigative function.
9. The method of claim 8, wherein the second investigative module comprises data including an identifier of an investigative function to run, and one or more parameters for the at least one investigative function.
10. The method of claim 8, wherein the second investigative module comprises one or more functions, executable by the first investigative module.
11. The method of claim 1, wherein at least one of the data fingerprints comprises a hash of a subset of the investigation data or a hash of the entirety of the investigation data.
12. The method of claim 1, wherein the data fingerprints include hashes representing the investigation data or subset thereof.
13. The method of claim 1, wherein the at least one investigative module performs at least one of: (i) hashing of the at least two data form attributes of the reference investigation data to produce the reference data fingerprint, and (ii) hashing of the at least two data form attributes of the comparison investigation data to produce the comparison data fingerprint.
14. The method of claim 1, wherein the investigation system stores data records, the data records including a record including: (i) at least one of the data fingerprints, (ii) an identifier of the corresponding host computer, and (iii) an identifier of the at least one investigative module run to produce the at least one of the data fingerprints.
15. The method of claim 1, wherein the computer investigations include using multiple different investigative modules performing corresponding second investigation functions, each of said multiple different investigative modules returning corresponding second data fingerprints that are collected by the investigation system.
16. The method of claim 1, wherein the investigation system produces a single data fingerprint representing the investigation data returned by multiple different investigative modules, the single data fingerprint being a representation of the combination of all, or part, of the investigation data returned by the multiple different investigative modules.

17. The method of claim 1, wherein the at least one investigative module is configured to generate multiple reference data fingerprints, each of the multiple reference data fingerprints corresponding to a different subset of the reference investigation data.
18. The method of claim 17, wherein the reference data fingerprint is a first data fingerprint, wherein the investigation system stores the multiple reference data fingerprints for each subset of the reference investigation data, including at least: (i) the first reference data fingerprint of a first subset of the reference investigation data, and (ii) a second reference data fingerprint of a second subset of the reference investigation data, wherein said first and second subsets are different.
19. The method of claim 18, wherein the second subset may be a superset of the first subset.
20. The method of claim 19, wherein the subsets of the reference investigation data correspond to at least one of: different data forms; and different sets of data form attributes.