US 20080016570A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2008/0016570 A1

Capalik (43) Pub. Date: Jan. 17, 2008

(54) SYSTEM AND METHOD FOR ANALYZING UNAUTHORIZED INTRUSION INTO A COMPUTER NETWORK

(76) Inventor: **Alen Capalik**, Los Angeles, CA (US)

Correspondence Address:
MORGAN, LEWIS & BOCKIUS, LLP.
2 PALO ALTO SQUARE
3000 EL CAMINO REAL
PALO ALTO, CA 94306 (US)

(21) Appl. No.: **11/788,795**

(22) Filed: **Apr. 20, 2007**

Related U.S. Application Data

(63) Continuation-in-part of application No. 11/488,743, filed on Jul. 17, 2006.

Publication Classification

(51) **Int. Cl.**
G06F 12/14 (2006.01)

(52) **U.S. Cl.** **726/23**

(57) **ABSTRACT**

The method analyzes unauthorized intrusion into a computer network. Access is allowed through one or more open ports to one or more virtualized decoy operating systems running on a hypervisor operating system hosted on a decoy network device. This may be done by opening a port on one of the virtualized decoy operating systems. A network attack on the virtualized operating system is then intercepted by an introspection module running on the hypervisor operating system. The attack-identifying information is communicated through a private network interface channel and stored on a database server as forensic data. A signature-generation engine uses this forensic data to generate a signature of the attack. An intrusion prevention system then uses the attack signature to identify and prevent subsequent attacks. A web-based visualization interface facilitates configuration of the system and analysis of (and response to) forensic data generated by the introspection module and the signature generation engine, as well as that stored in the processing module's relational databases.

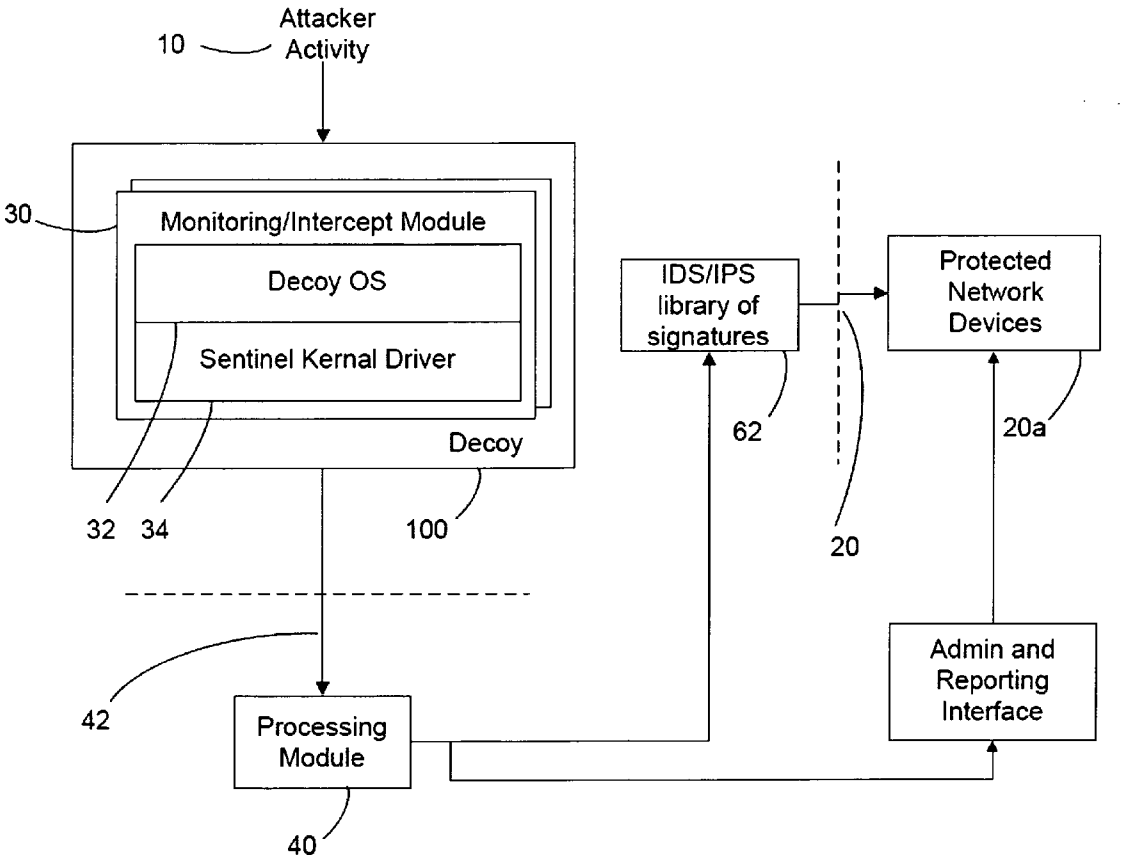


Figure 1

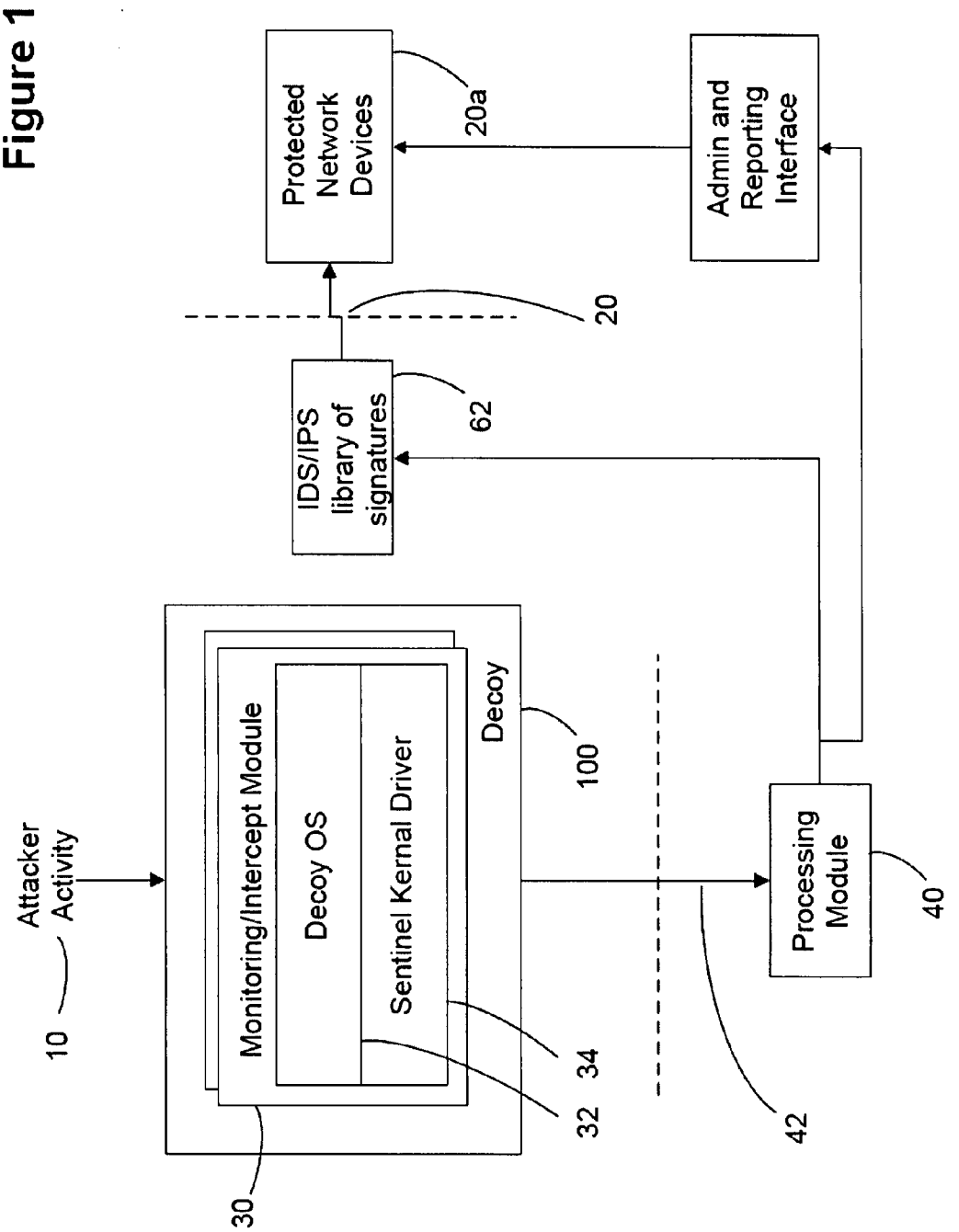

Protected Network Devices
20a
Admin and Reporting Interface
20
IDS/IPS library of signatures
62
Attacker Activity
10
Monitoring/Intercept Module
Decoy OS
Sentinel Kernal Driver
Decoy
100
30
32
34
42
Processing Module
40

Figure 2

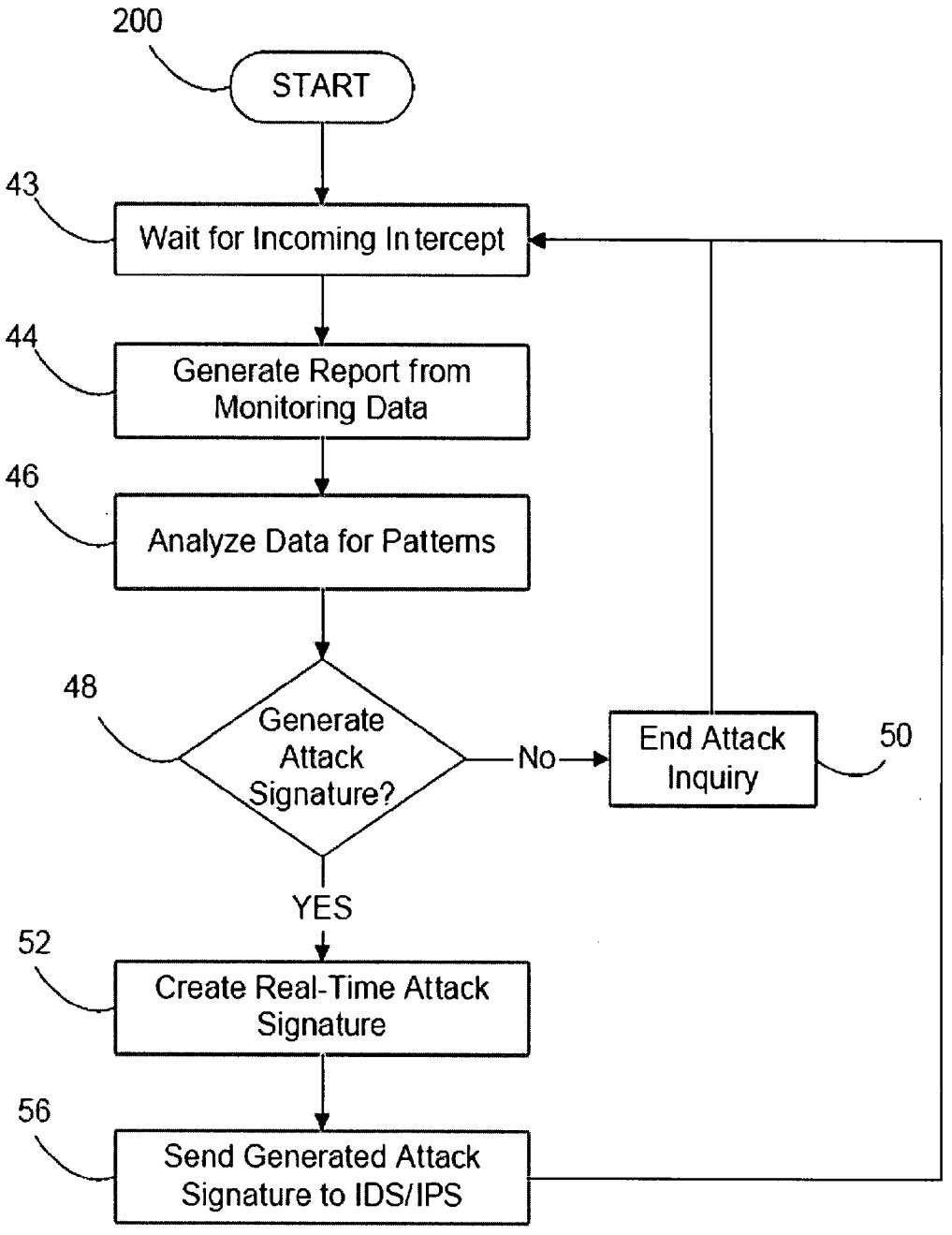

200
START
43
Wait for Incoming Intercept
44
Generate Report from Monitoring Data
46
Analyze Data for Patterns
48
Generate Attack Signature?
No
End Attack Inquiry
50
YES
52
Create Real-Time Attack Signature
56
Send Generated Attack Signature to IDS/IPS

Figure 3

When first run W32/Rbot-CBQ copies itself to <System>\msprexe.exe. ← 300

The following registry entries are created to run msprexe.exe on startup:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run ← 301
Windows Management System
msprexe.exe

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run ← 302
Windows Management System
msprexe.exe

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices
Windows Management System
msprexe.exe

Registry entries are set as follows:

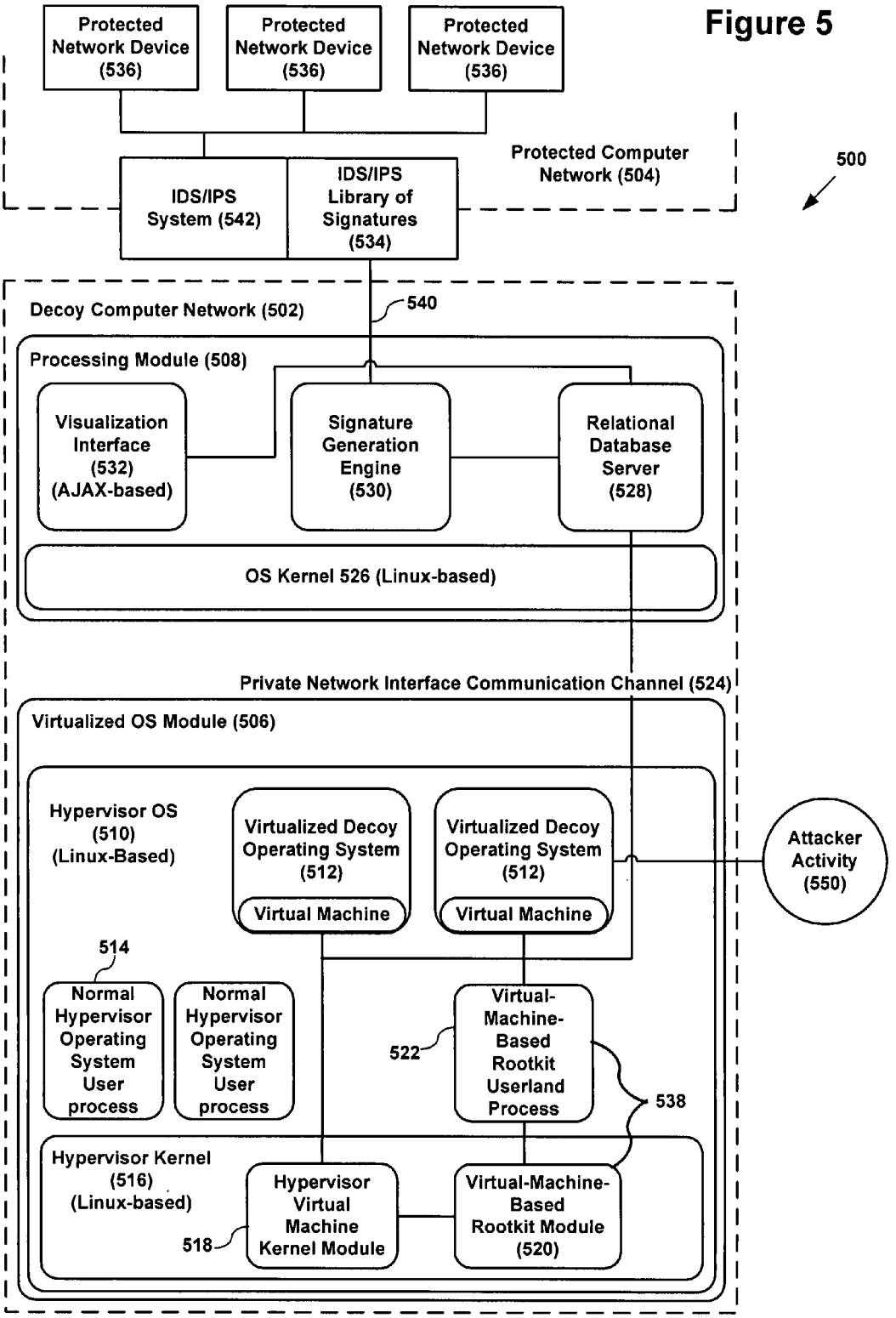
HKLM\SOFTWARE\Microsoft\Ole
EnableDCOM
N

HKLM\SYSTEM\CurrentControlSet\Control\Lsa
restrictanonymous
1

Figure 4

```
<item type="STRING.TCP">                ← 400
 <struct name="STRING.TCP">
  <var name="version" protected="true">4.0</var>
  <array name="signatures">
   <entry dontDelete="true" nda="false" sfr="85">
    <var name="SigName" default="RBOT.CBQ Worm Activity" protected="true" />
    <var name="SIGID" default="5571" protected="true" />
    <var name="SubSig" default="0" protected="true" />
    <var name="AlarmSeverity" default="high" />
    <var name="Enabled" default="True" />
    <var name="EventAction" default="alarm">alarm|reset</var>
    <var name="SigVersion" default="S185" />
    <var name="SigStringInfo" default="RBOT.CBQ" />
    <var name="AlarmThrottle" default="Summarize" />
    <var name="Direction" protected="true" default="ToService" />
    <var name="MinHits" default="1" />
    <var name="Protocol" default="TCP" />
    <var name="RegexString" protected="true"
default="\xb5\x71\xc4\x43\x1e\x34\xce\x3a\x18\x37\xef\xc3\x22\xa1\x20\x1f\x6c\x10\x8f\x6
8\x3d\xf4\xef\x51\x92\xf9\x4f\xbc\x46\xe9\x05\xdc" />
    <var name="ServicePorts" default="445" />
    <var name="StorageKey" default="STREAM" />
    <var name="SummaryKey" default="Axxx" />
    <var name="ThrottleInterval" default="15" />
   </entry>
  </array>
 </struct>
</item>
```

Figure 5


Protected Network Device (536)
Protected Network Device (536)
Protected Network Device (536)
Protected Computer Network (504)
500
IDS/IPS System (542)
IDS/IPS Library of Signatures (534)
Decoy Computer Network (502)
540
Processing Module (508)
Visualization Interface (532) (AJAX-based)
Signature Generation Engine (530)
Relational Database Server (528)
OS Kernel 526 (Linux-based)
Private Network Interface Communication Channel (524)
Virtualized OS Module (506)
Hypervisor OS (510) (Linux-Based)
Virtualized Decoy Operating System (512)
Virtual Machine
Virtualized Decoy Operating System (512)
Virtual Machine
Attacker Activity (550)
514
Normal Hypervisor Operating System User process
Normal Hypervisor Operating System User process
522
Virtual-Machine-Based Rootkit Userland Process
538
Hypervisor Kernel (516) (Linux-based)
518
Hypervisor Virtual Machine Kernel Module
Virtual-Machine-Based Rootkit Module (520)

SYSTEM AND METHOD FOR ANALYZING UNAUTHORIZED INTRUSION INTO A COMPUTER NETWORK

[0001] This is a Continuation-In-Part, and claims priority to and the benefit of: U.S. Ser. No. 11/488,743, entitled "Decoy Network Technology With Automatic Signature Generation for Intrusion Detection and Intrusion Prevention Systems" filed on Jul. 17, 2006, the entire disclosure of which is incorporated herein by reference.

FIELD

[0002] The invention relates to the field of methods and systems for protecting computer networks and is more particularly, but not by way of limitation, directed to decoy network technology with automatic signature generation for intrusion detection and intrusion prevention systems.

BACKGROUND

[0003] Computer networks typically interface with the Internet or other public computer systems and are thus vulnerable to attacks, unwanted intrusions and unauthorized access. One threat to networks is the so-called zero-day attack that exploits security vulnerabilities unknown to the system operators.

[0004] Conventional network security systems include a firewall that generally prevents unauthorized access to the network or its computers. Conventional systems also include intrusion detection systems (IDS) and intrusion prevention systems (IPS) that typically contain a library of signatures of malware payloads, which enable them to detect those defined exploits attempting to access production systems. When a connection is attempted to a network port, the IDS or IPS examines the low-level IP data packets and compares them to its library of signatures for a match. When a match is identified the IDS or IPS provides notification of the match.

[0005] The problem lies in the static nature of the conventional IDS and IPS signatures coupled with the ability of determined attackers to launch new undefined or zero-day automated attacks to gain access to the network. While an intrusion prevention system (IPS) equipped with behavioral signatures providing the ability to capture behavioral patterns offers a higher level of protection, these have similar drawbacks in that behavioral signatures are still static in nature and limited in their ability to stop zero-day attacks.

[0006] Still another type of network security systems utilizes a honeynet arrangement to attract and then trap a suspected attacker. A honeynet is made up of two or more honeypots on a network. Such measures typically are made up of a computer, data or network site that appears to be part of the network and appears to be one or more valuable targets, but which is actually an isolated component located away from production networks. These are typically passive measures effective against spammers and other low-level attacks. Such systems typically run emulated operating systems and services and are generally not useful against sophisticated attackers who can detect and effectively avoid the honeynet, never unloading their zero-day attack or payload for the honeynet to capture and analyze. Also, if the conventional honeynet configuration is not sufficiently separated from the network system, an attacker can use the honeynet to gain access to the network. Examples of emulated or software based honeypots include "honeyd" which is a GPL licensed daemon that is utilized to simulate network structures. Another example of emulated software based honeypots include "mwcollect" and "nepenthes" which are also released under the GPL license and which are utilized to collect malware. The "mwcollect" and "nepenthes" packages extract information on obtaining the malware binaries from the exploit payload.

[0007] Because each of the problems and limitations discussed above exist in the prior art devices and systems, there is a need for methods and systems that adequately protect networks from new and undefined attacks and that allow for real-time updates to a network's library of attack signatures.

SUMMARY

[0008] One or more embodiments of the invention are directed to an improved method and system for protecting computer networks. In one embodiment, the invention comprises a modular decoy network appliance, which runs fully functional operating systems on client hardware modules. The modular arrangement comprises front-end fully functional operating system modules and a separate processing back-end module.

[0009] The front-end presents a standard fully functional operating system, such as Windows® or a flavor of Linux®, or Sun Microsystems Solaris® that returns a standard operating system fingerprint when it is scanned by tools that attackers typically use to identify vulnerable systems. The attacker is thus lured into accessing the identified operating system and running custom or known exploits on that system.

[0010] The front-end module includes a sentinel kernel driver (or a more generalized executable module) that is hidden from system scanners as it is removed from kernel module listings or registry in Windows. Thus, the kernel does not indicate the sentinel kernel driver is running. The sentinel kernel driver monitors connections to the operating system as well as activity on the operating system and activity on services running on the operating system. When an attacker connects to a port, the sentinel kernel driver captures the data coming through the socket. Generally all relevant data coming through the socket is captured. In most cases this means whatever data is received as part of an incoming attack is captured by the sentinel driver. Captured data is sent as a slew of common UDP packets to the back end processing module over the fabric network connection separate from the vulnerable front-end modules. In this manner, there is no way for the intruder to know that his or her communications with the operating system are being analyzed.

[0011] The captured data, which contains the attack-identifying information, is sent to the back-end or processing module though the backplane fabric of the appliance using Layer 2 Ethernet communication protocol. The processing module is separate and independent from the client operating system modules and communicates the processed information to security administrators through a network port connected to the private and secure VLAN. Unbeknownst to the intruder, the exploit is thus captured, transferred and analyzed.

[0012] With the received data, the processing module generates a report of the attack. The report consists of user-friendly information that paints a picture of the attack for a system administrator. This may include information on which sockets were accessed, what happened at a particular socket, the key strokes entered or bytes transferred to the port, what files were transferred, registry changes, how the attack was run, what happened on the primary network, on its servers or how the network services were affected. The report may also include information on the location of the attacker or the attacker's service provider. Graphical representations of key information and interactive mapping of the attack locales by region or country may be utilized in one or more embodiments of the invention.

[0013] The processing module is used to generate an attack signature by analyzing all the data passed through the socket. The signature is generated by analyzing the attack payload including the keystrokes or transferred bytes and any files uploaded to the client operating system of an ASCII or binary nature. The files uploaded are assumed to be of a malicious nature created to deliver a malicious payload in the form of a compiled program or an interpreted script. In the event that no malicious files are uploaded to the operating system, the signature generation engine analyzes all the keystrokes or bytes delivered through the socket and creates a pattern signature which when applied to an IDS or IPS system, enables the IDS or IPS systems to detect the attack if repeated on production systems. Once generated, the attack signatures can be viewed by a system administrator to determine the appropriate course of action. The system administrator can instruct the signature to be uploaded to the intrusion detection system (IDS) or intrusion prevention system (IPS) for the protected network where it is added to the IDS's or IPS's library of signatures to protect production systems. In one or more embodiments of the invention, the signature may be uploaded or saved in a third party system that maintains all known exploits. In this manner, other systems may be notified through secure channels of an impending threat. For example, by transferring the signature to a centralized server that communicates with multiple installations, the intruder may be thwarted before attacking other systems in other companies.

[0014] A production network's library of signatures can be updated in real-time as the attacker modifies its illicit activity or a new attack is launched. The embodiment can also maintain a database of any and all attack signatures generated. Other and further advantages will be disclosed and identified in the description and claims and will be apparent to persons skilled in the art.

[0015] Another embodiment provides a system and method for analyzing unauthorized intrusion into a computer network. Access is allowed through one or more open ports to one or more virtualized decoy operating systems running on a hypervisor operating system hosted on a decoy network device. This may be done by opening a port on one of the virtualized decoy operating systems. A network attack on the virtualized operating system is then intercepted by a virtual-machine-based rootkit module running on the hypervisor operating system. The attack-identifying information is communicated through a private network interface channel and stored on a database server as forensic data. A signature generation engine uses this forensic data to generate a signature of the attack. An intrusion prevention system then uses the attack signature to identify and prevent subsequent attacks

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] FIG. **1** illustrates a block diagram of an embodiment of the system;

[0017] FIG. **2** illustrates a flow chart of an embodiment of the processing that occurs on processing module **40**;

[0018] FIG. **3** illustrates a human readable summary of an example attack;

[0019] FIG. **4** illustrates an XML formatted attack signature generated from the attack summarized in FIG. **3** for transmittal to an IDS or IPS; and

[0020] FIG. **5** illustrates a block diagram of another embodiment of the system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0021] The following descriptions of embodiments of the invention are exemplary, rather than limiting, and many variations and modifications are within the scope and spirit of the invention. Although numerous specific details are set forth in order to provide a thorough understanding of the present invention, it will be apparent to one of ordinary skill in the art, that embodiments of the invention may be practiced without these specific details. In other instances, well-known features have not been described in detail in order to avoid unnecessarily obscuring the present invention.

[0022] One or more embodiments of the invention are directed to an improved method and system for protecting computer networks. One embodiment is illustrated in FIG. **1**, which illustrates attacker activity **10** directed at protected computer network **20**. As in a typical attack, attack **10** is scanning for an open port on computer network **20** in an attempt to make a connection and then access one or more protected network devices **20***a* on network **20**.

[0023] Attack **10** is monitored by decoy **100** that includes at least one monitor/intercept module **30**. Monitor/intercept module **30** comprises fully functioning decoy operating system **32** that monitors each of the access ports for network **20**. Any operating system may be used as decoy operating system **32** including Windows®, Sun Microsystems Solaris® or any version of Linux® known to persons skilled in the art. All known operating systems are within the scope of the present invention. FIG. **1** shows one monitoring/intercept module **30** in the foreground, however any number of homogeneous or heterogeneous monitoring/intercept modules may be utilized (shown as a stack behind monitor/intercept module **30**). For example, in one embodiment of the invention a Windows® monitoring/intercept module **30** and LINUX® monitoring/intercept module **30** may be employed. There is no limit to the number of monitoring/intercept modules that may be utilized in the system and other embodiments may employ homogeneous decoy operating systems **32** that are of the same or of different versions. Monitoring/intercept module **30** also includes sentinel kernel driver **34** which will be described in further detail below. Protected network devices **20***a* are accessed through IDS/

IPS with Library of Signatures **62** in one or more embodiments of the invention. The system also includes processing module **40** for obtaining and analyzing exploits.

[**0024**] When attack **10** connects to an access port of network **20**, the fully functional decoy operating system **32** intercepts the connection and returns a standard operating system fingerprint. For example when connecting to an address that does not exist on protected network **20**, decoy **30** may be configured to respond to any such incorrect address since the connection is assumed to be malicious as there is no hardware on protected network **20** at that address. The response may be configured to utilize any existing hardware module having a given operating system and version within monitoring/intercept module **30**. For example, an FTP port access for Windows® may return a particular character sequence that is different than an FTP response for LINUX®. An FTP access to a Windows® port for example may return a response ">ftp: connect: Connection refused". This characters sequence may be slightly different on LINUX® and hence allows the intruder to determine what type of operating system is at a particular network address. In addition, different versions of Windows® may respond with slightly different character sequences which allows the intruder to determine the specific version of the operating system or to determine a possible range of versions for the responding operating system. The instigator of attack **10** is thus lured into accessing decoy **100**, which includes monitor/intercept module **30**, and running custom or known exploits for the observed operating system. When attacker activity proceeds to interact with decoy **100**, the attacker provides decoy **100** with the data used to obtain control of decoy **100**, which is recorded and analyzed without knowledge of the attacker.

[**0025**] All scans by attack **10** receive real-world operating system information, thus leading the instigator of the attack **10** to believe that there is a potentially vulnerable system responding and thus luring attack **10** into communicating with monitor/intercept module **30**. Since real hardware is utilized, the attacker is attacking an actual physical system and thus has no idea that the system is actually an instrumented honeypot that monitors the attackers every move.

[**0026**] Monitor/intercept module **30** includes sentinel kernel driver **34**. In one embodiment, sentinel kernel driver **34** is a combination of custom root-kit code that on Windows® based operating systems removes pointers from Microsoft® client/server runtime server subsystem (CSRSS.exe). This coupled with removing sentinel kernel driver **34** from the Windows® registry effectively hides sentinel kernel driver **34** and all its drivers from attack **10**. On Unix® based operating systems, the kernel pointers are removed making the kernel unable to link to a running process, effectively hiding sentinel kernel driver **34** and all its libraries from attack **10**. Sentinel kernel driver **34** monitors all data coming through the socket and is derived from an open source code, such as libpcap, known to persons skilled in the art.

[**0027**] When an attacker connects to a port, and begins interacting with decoy operating system **32**, sentinel **34** monitors and captures information from the connection including port numbers, data streams, keystrokes, file uploads and any other data transfers.

[**0028**] The captured information, or attack-identifying information, is then sent for processing to processing module **40** as illustrated in FIG. **1**. Processing module **40** may optionally include a sentinel server that receives information from the sentinel kernel driver and deposits the information in a database for later analysis. In one embodiment, the monitor/intercept module **30** is a front-end module or series of modules and the captured data is sent to processing module **40** though the backplane of the appliance or appliances through a layer 2 Ethernet communications link not available to the attacker such as an IP connection or any other hardware dependent custom communication protocol known to persons skilled in the art. Processing module **40** is part of a secure and separate administrative network **42**. In one or more embodiments the signature may be sent from the back end processing module **40** to IDS/IPS **62** through a second network connection which is used by the processing module **40** to directly interact with IDS/IPS **62**. The sentinel kernel driver may utilize replay functionality to replay the attacks on the operating system in reverse to clean up the operating system to its pre-attack state. In this manner, the attack can be thwarted and the operating system thus does not become a tool of the hacker.

[**0029**] As shown in FIG. **2**, processing starts at **200** and waits for activity from sentinel kernel driver **34** at step **43**. In step **44**, processing module **40** generates a report of the attack that includes attack-identifying information (See FIG. **3**). This report is for the use and review by a system administrator who is responsible for administering protected network **20**. The attack may contain one or more data transfers or keystrokes for example, which are analyzed at step **46**. By observing whether the attacker is successful in interacting with the system, i.e., if the system is responding in a manner that shows that the attacker has gained access, then determination whether to generate an attack signature is made at step **48** and the attack signature is generated at step **52** (See FIG. **4**). If the attacker for example is unsuccessful at gaining access or if there is no data transfer for example, then the attack inquiry may be ended at step **50**. Any generated attack signature is sent to the IDS/IPS at step **56** and processing continues at step **43**.

[**0030**] In one embodiment of the invention, the report is written, and is displayed in an web-based visualization interface and can include information about which sockets were accessed by attack **10**, what happened at a particular socket, the key strokes entered or data transferred, what files were transferred, how the attack **10** was run, what happened on monitor/intercept module **30** and how decoy operating system **32** and any related network services were affected. The report may also include information on the location of the instigator of attack **10** or the service provider used for attack **10**. Graphical representations of key information and interactive mapping of attack locales by region or country may also be included in the report.

[**0031**] In step **46**, the attack-identifying information is analyzed for known attack patterns and non-standard patterns such as repeating binary patterns, keystroke patterns, downloaded daemons or errors such as buffer overflow attempts. By observing the operations performed on decoy operating system **32** the attack may be categorized and analyzed to determine for example how an attack gains control of decoy operating system **32**. Any method of analyzing the incoming data such as binary matching, neural network matching or keyword matching or any other method

of matching attack-identifying information is in keeping with the spirit of the invention.

[0032] In step **48**, a decision is made as to whether to generate an attack signature. If no harmful operations occurred as a result of attack **10** or when no known attack patterns are found, then no further attack inquiry would be needed as shown in step **50**. The processing module **40** can then take on the next input of captured information from the monitor/intercept module **30**.

[0033] If a determination is made that attack signature generation is warranted, an attack signature is generated as illustrated in step **52**. Processing module **40** may generate a signature whenever data is found to be transferred through the socket in one or more embodiments of the invention. Alternatively, if the attack signature already exists or if the data transfer is of a nature that indicates probing rather than attack, then the attack signature may not be generated. For example, processing module **40** may not generate a signature when it is found that no data has been transferred through the socket even though the socket may have been opened and closed without data transfer. Once the attack signature is generated, the signature can be reviewed by the system administrator who decides to send the attack signature, shown in step **56**, to the intrusion detection system (IDS) or intrusion prevention system (IPS) for the protected network **20** through a standard network connection including a wireless connection that is generally not sent on protected network **20** or any other network that the attacker may observe. This is accomplished by applying the generated attack signature to the IDS/IPS library of signatures to update the information contained in the library of signatures to prevent the attacker from accessing the primary network with a zero-day attack.

[0034] Embodiments of step **56** may save the generated attack signatures in a database for future use or further analysis by system administrators. The signatures may also be sent to a proprietary global database of attack signatures for further analysis. Any IDS/IPS may be utilized in one or more embodiments of the invention. Existing IDS/IPS systems for example may be interfaced with in order to integrate with existing solutions.

[0035] FIG. **3** illustrates a human readable summary of an example attack. Line **300** shows that the file "msprexe.exe" is copied into the "System" directory. Line **301** shows a first registry entry created by the attack. Line **302** shows a second registry entry created by the attack. Any other changes to the system may be shown, as part of the attack-identifying information and the information shown in FIG. **3** is exemplary only.

[0036] FIG. **4** illustrates an XML formatted attack signature generated from the attack summarized in FIG. **3** for transmittal to an IDS or IPS. XML block **400** includes tags that define the attack signature in the format of the particular IDS or IPS. Any tags used by any IDS or IPS are in keeping with the spirit of the invention and the tags shown in FIG. **4** are exemplary only. For example any ports, protocols, severity levels, alarm levels, signature name or any other quantity may be utilized to inform an IDS or IPS of an attack signature.

[0037] Another embodiment of a system for analyzing and preventing unauthorized intrusion into a computer network is shown in FIG. **5**. This embodiment is directed to an improved method and system for analyzing unauthorized intrusion into a decoy computer network, the analysis of which is used to prevent unauthorized access into a protected computer network. An embodiment of such a system is illustrated in FIG. **5**, while the method remains as shown in the flowchart in FIG. **2** above.

[0038] The system **500**, as shown in FIG. **5**, includes a decoy computer network **502** and a protected computer network **504**, each comprising one or more separate computing devices. The decoy computer network **502** includes a virtualized operating system module **506** for monitoring the decoy network **502**, and a processing module **508** for obtaining, analyzing, and responding to exploits.

[0039] These modules may be hosted on the same computing device or on separate computing devices. However, for ease of explanation, these modules will be described below as being hosted on separate computing devices. Furthermore, although not shown, one skilled in the art will appreciate that each of these computing devices may include one or more processors, input/output devices, communication circuitry, power sources, memory (both physical, e.g., RAM, and disks, e.g., hard disk drives), and any other physical hardware necessary for hosting and running the aforementioned modules. In some embodiments, the modules **506** and **508** are as present in physical memory once the system has been booted and is operational.

[0040] The virtualized operating system module **506** includes a hypervisor operating system **510** (also known as a virtual machine monitor operating system) that provides a virtualization platform that allows multiple virtual operating systems to be run on a host computing device at the same time. In some embodiments, the hypervisor operating system **510** is a LINUX-based system. One or more fully-functioning "guest" virtualized operating systems **512** are run on the hypervisor operating system **510** at a level above the hardware. As will be described in detail below, these virtualized operating systems **512** act as decoy operating systems to attract attacker activity **550**. Any operating system may be used as guest decoy operating system **512**, including but not limited to WINDOWS, SUN MICROSYSTEMS, SOLARIS, or any version of LINUX known to persons skilled in the art, as well as any combination of the aforementioned. It should be appreciated that all known operating systems are within the scope of the present invention. There is also no limit to either the number of virtualized guest decoy operating systems **512** or the number of virtualized guest operating system modules **506** that may be utilized.

[0041] Also running on the hypervisor operating system **510** are normal hypervisor operating system userland processes **514**. The hypervisor operating system **510** includes a hypervisor kernel **516**, which in some embodiments is also Linux-based. The hypervisor kernel **516** is that part of the hypervisor operating system **510** that resides in physical memory at all times and provides the basic services to the hypervisor operating system **510**. The hypervisor kernel **516** is the part of the operating system that activates the hardware directly or interfaces with another software layer that, in turn, drives the hardware. The virtualized decoy operating systems **512** access the physical memory assigned to them by the hypervisor operating system via the hypervisor kernel **516**.

[0042] The hypervisor kernel **516** includes a hypervisor virtual machine kernel module **518** that supports virtualization of the "guest" decoy operating systems **512**. The hypervisor kernel **516** also includes virtual-machine-based rootkit module **520** coupled to the hypervisor virtual machine kernel module **516**. The virtual-machine-based rootkit module **520** is a set of software tools that conceal running processes, files or system data from the virtualized decoy operating systems **512**. As described in further detail below, the virtual-machine-based rootkit module **520** is part of introspection module **538**, which performs introspection into the physical memory segments assigned to each of the virtualized decoy operating systems **512**.

[0043] Virtual-machine-based rootkit userland processes **522** run on top of the virtual-machine-based rootkit module **520**. Together, the rootkit module **520** and its associated userland processes **522** constitute the system's introspection module **538** (described further below). Virtual-machine-based rootkit userland processes **522** also pass data from the introspection module **538** to the processing module **508**.

[0044] In use, attacker activity **550** is directed at the decoy computer network **502** through one or more ports of each of the virtualized decoy operating systems **512** that are left open as a gateway for attacker activity **550**. For example, the decoy network **502** can be configured to respond to connection attempts made at network addresses that do not exist on the protected network **504**. Connections to these non-existent network addresses are assumed to be malicious, since no production hardware exists on the protected network **504** at these addresses. Decoys **512** (in the form of a virtualized operating system) may be configured to respond to any such non-existent network address. As in a typical attack, the attacker activity **550** scans for an open port, ostensibly in an attempt to make a network connection and then access one or more computing devices on the protected computer network **504**. When the attacker activity **550** scans for open ports at non-existent network addresses, however, the attacker is presented with a virtualized decoy operating system **512** instead.

[0045] When the attacker activity **550** connects to a virtualized decoy operating system **512** through an open port, the attacker sees a fully-functional standard operating system fingerprint. Since the virtualized operating system module **506** can be configured to present any operating system as a fully-functional virtualized decoy **512**, responses to connection requests from attacker activity **550** are guaranteed to be authentic for the operating system running on that decoy. For example, an FTP port access request for WINDOWS may return a specific character sequence that differs from an FTP response for LINUX. Similarly, an FTP access request to a WINDOWS port may return a response ">ftp: connect: Connection refused." This character sequence may be slightly different from that generated by LINUX. Further, different versions of WINDOWS may respond with slightly different, version-specific character sequences. Since attackers often use these sequences to identify what type of operating system is at a particular network address and the version (or range of possible versions) for that operating system, the fact that virtualized decoy operating systems **512** generate authentic responses makes them realistic decoys and encourages intruders to access them. The instigator of the attack **550** is thus lured into accessing the decoy **512**, which is overseen by the hypervisor operating system **510** running on the hardware-based, virtualized operating system module **506**. Attacker activity **550** may then initiate custom or known exploits for the observed operating system. When the attacker activity **550** proceeds to interact with the decoy **512**, the attacker provides the decoy **512** with the data used to obtain control of the decoy **512**. These data are recorded and analyzed without the knowledge of the attacker, as described further below.

[0046] All scans by the attacker activity **550** receive real-world operating system and service information, leading the instigator of the attack **550** to believe that there is a potentially vulnerable system responding. The attacker is thus lured into communicating with virtualized operating system module **506** and its virtualized decoy operating systems and services. Since real hardware is utilized, the attacker is essentially attacking an actual physical system and, therefore, cannot tell that the system is actually an instrumented honeypot that monitors the attacker activity **550** from the introspection module **538** described below.

[0047] As described above, the virtualized guest operating system module **506** includes the virtual machine-based rootkit module **520** and its associated userland processes **522**. Since both the virtual machine-based rootkit module **520** and its associated userland processes **522** run completely outside the virtualized decoy operating systems **512**, they remain hidden from the instigator of the attack, with no discoverable impact on the decoy operating systems' **512** performance. In one embodiment, the virtual machine-based rootkit module **520** and its associated userland processes **522** constitute an introspection module **538** (also known as a virtual machine-based memory introspection analysis tool) that monitors and introspects into the virtualized decoy operating systems' memory segments. This occurs from within the hypervisor operating system **510**. The introspection module **538** introspects and gathers information on any virtualized operating system supported by the hypervisor operating system **510**.

[0048] The introspection module **538** comprising the virtual-machine-based rootkit module **520** and its associated userland processes **522** examines the memory assigned to virtualized decoy operating systems **512** in order to acquire low-level data about the interaction between the decoy operating systems and attack activity **500**. The introspection module **538** examines the memory of virtualized decoy operating systems **512** by means of three functional components: a code region selector, a trace instrumentor, and a trace analyzer. Regular expressions (also known as 'regex') are used throughout the process to identify, describe, and profile the contents of the virtualized decoy's memory segments. The code selector identifies regions of code in memory that are of interest for further introspection. Regions of interest may include, but are not limited to, system calls, the arguments of system calls, the returns of system calls, device and memory input-output, driver information, library calls, branching information, instruction pointer jumps, and raw network information. The instrumentor copies the memory traces of interest identified by the code selector and then profiles and instruments them. The trace analyzer takes the instrumented traces and uses them to replay the memory behavior of the decoy operating system **512**. In this manner, the introspection module **538** examines the contents of the decoy operating systems' **512** memory

segments in an instrumented context that generates and retrieves forensic data for analysis by the processing module **508**.

[0049] When an attacker connects to a network port and begins interacting with a virtualized decoy operating system **512**, the introspection module **538** monitors and captures information from the connection, including port numbers, data streams, file uploads, keystrokes, ASCII or binary files, malicious payloads, memory manipulation attempts, and any other data transfers or malicious attempts.

[0050] The captured information, containing attack-identifying information, is then sent from the introspection module **538** to the processing module **508** by means of a virtual machine-based rootkit userland process **522**.

[0051] The processing module **508** includes an operating system kernel **526**, which in some embodiments is also LINUX based. The processing module **508** also includes a database, such as a relational database server **528**, and a signature-generation engine **530**. In some embodiments, the signature-generation engine **530** communicates with the introspection module **538** over a private network interface communications channel **534** and accepts custom-formatted protocol packets named BAT (Blade Activity Transfer). The private network interface communications channel **524** may be a persistent Layer 3 TCP socket communications link that cannot be seen or accessed by the attacker (such as an IP connection or any other hardware-dependent custom communication protocol known to persons skilled in the art). Thus, the processing module **508** is part of a secure and separate administrative network.

[0052] In use, the introspection module **538** captures (through introspection) attack information. The attack information is then communicated through the private network interface channel **524** and stored on the relational database server **528** as forensic data for later analysis. The signature-generation engine **530** then uses this forensic data to generate a signature of the attack. The entire process from attack detection through signature generation may occur automatically, i.e., without any human intervention, at a timescale ranging from nearly immediate to several minutes. The intrusion prevention system (described below) uses the attack signature to identify and prevent subsequent attacks.

[0053] The protected computer network **504** includes an IDS/IPS library of signatures **534** and an IDS/IPS system **542** coupled to multiple protected network devices **536**. Suitable IDS/IPS systems **542** include Cisco Systems' IPS 4200 Series, Juniper's IDP **200**, and Enterasys' Dragon IDS Network Sensor.

[0054] In one or more embodiments, the signature may be sent from the back-end processing module **508** to the intrusion detection and/or prevention (IDS/IPS) signature library **534** through a second network connection **540**, which is used by the processing module **508** to directly interact with the IDS/IPS system **542**. The virtual-machine-based rootkit module **520** may easily clean the virtualized decoy operating system **512** at any time by removing the running system image of the compromised virtualized decoy operating system and replacing it with a pre-attack system image. Thus the virtual-machine-based rootkit module **520** can cleanse or reset the virtualized decoy operating system of any malicious software or payload, removing the possibility that attacker(s) can use that virtualized decoy operating system **512** for further attacks on other networks. In this manner, the attack can be thwarted, and the operating system does not become a tool of the attacker(s). This procedure may also be automated, i.e., may occur without further human intervention.

[0055] As shown in FIG. **2**, processing starts at Step **200** and waits for activity from the introspection module **538** at Step **43**. At Step **44**, the processing module **508** generates a report of the attack that includes attack-identifying information (See FIG. **3**). This report is for review and use by a system administrator responsible for the security of a protected network **504**. The attack may contain, but is not limited to, one or more data transfers or keystrokes, which are analyzed at Step **46**. By observing whether the attacker is successful in interacting with the system (i.e., if the system is responding in a manner that shows that the attacker has gained access), a determination can be made at Step **48** as to whether an attack signature should be generated, and the attack signature is created at step **52** (See FIG. **4**). If the attacker, for example, is unsuccessful at gaining access, or if there is no data transfer, the attack inquiry may be ended at Step **50**. Any attack signature generated is sent to the IDS/IPS signature library **534** at Step **56**, and processing continues at Step **43**.

[0056] In one embodiment of the invention, the report of the attack is written and then displayed via a visualization interface **532** and can include information about which sockets were accessed by the attack **550**, what happened at a particular socket, the keystrokes entered or data transferred, what files were transferred, how the attack **550** was run, what happened on the virtualized operating system module **506**, and how the virtualized decoy operating systems **512** running on the hypervisor operating system **510** and any related network services were affected. In some embodiments, the visualization interface **532** is AJAX- and/or FLASH-based. The report may also include information on the location of the instigator of the attack **550** or the service provider used for the attack. Graphical representations of key information and interactive mapping of attack locales by region or country may also be included in the report. The visualization interface may also be used to analyze, configure, and automate the system's response to attack activity **550** on timescales ranging from near-immediate to several minutes from the initiation of an attack.

[0057] At Step **46**, the attack-identifying information is analyzed for known attack patterns as well as non-standard patterns, such as repeating binary patterns, keystroke patterns, downloaded daemons, or errors (such as buffer overflow attempts, malicious payloads attempting to execute arbitrary code on the system, memory overwriting attempts, stack attacks, and heap attacks). By observing the operations performed on the decoy operating system(s) **512**, the attack **550** may be categorized and analyzed to determine, for example, how an attack gained control of the decoy operating system(s) **512**. Any method of analyzing the incoming data such as binary matching, neural-network matching, keyword matching, or any other method of matching attack-identifying information is in keeping with the spirit of the invention. Pattern-matching techniques involving neural networks, for example, are characterized in Carl Looney's *Pattern Recognition Using Neural Networks. Theory and Algorithms for Engineers and Scientists* (Oxford University

Press USA, New York, N.Y., 1997) and Christopher Bishop's *Neural Networks for Pattern Recognition* (Oxford University Press USA, New York, N.Y., 1995), among other sources familiar to those skilled in the art.

[0058] At Step **48**, a decision is made as to whether to generate an attack signature. If no harmful operations occurred as a result of an attack, or when no known attack patterns are found, then no further attack inquiry would be needed (as shown at Step **50**). The processing module **508** may then take on the next input of captured information from the introspection module **538** running on the hardware-based, virtualized operating system module **506**.

[0059] If a determination is made that attack signature generation is warranted, an attack signature is generated as illustrated in Step **52**. In one or more embodiments of the invention, the processing module **508** may generate a signature whenever data is found to be transferred through the socket. Alternatively, if the attack signature already exists, or if the data transfer is of a nature that indicates probing rather than attack, then the attack signature may not be generated. For example, the processing module **508** may not generate a signature when it is found that no data has been transferred through the socket, even though the socket may have been opened and closed. The conditions under which the processing module **508** generates an attack can be configured and automated by an administrator. Once the attack signature is generated, the signature can be reviewed by the system administrator, who decides whether to send the attack signature (shown at Step **56**) to the intrusion detection system (IDS) or intrusion prevention system (IPS) for the protected network **504**. The attack signature is sent through a standard network connection or via a wireless connection and is generally sent on a private portion of the protected network **504** that the attacker cannot observe. The generated attack signature is thus applied to the IDS/IPS library of signatures **534**, thereby updating the information contained in the signature library and preventing the attacker from accessing the protected network **504**.

[0060] Embodiments of the invention may save the attack signatures created at Step **52** in a relational database server **528** for future use or analysis by system administrators. The signatures may also be sent to a proprietary global database of attack signatures for further analysis, storage, and distribution. Any IDS/IPS system may be utilized in one or more embodiments of the invention. The invention may be interfaced with existing IDS/IPS systems, for example to integrate it with existing solutions.

[0061] As explained above, FIG. **3** illustrates a human-readable summary of an example attack. Line **300** shows that the file "msprexe.exe" is copied into the "System" directory. Line **301** shows a first registry entry created by the attack. Line **302** shows a second registry entry created by the attack. Any other changes to the system may be shown as part of the attack-identifying information, and the information shown in FIG. **3** is exemplary only.

[0062] As explained above, FIG. **4** illustrates an attack signature generated from the attack summarized in FIG. **3** and formatted in XML for transmission to an IDS or IPS. XML Block **400** includes tags that define the attack signature in the format of the particular IDS or IPS. Any tags used by any IDS or IPS are in keeping with the spirit of the invention, and the tags shown in FIG. **4** are exemplary only. For example, any ports, protocols, severity levels, alarm levels, signature name, or any other quantity, may be utilized to inform an IDS or IPS of an attack signature.

[0063] While embodiments and alternatives have been disclosed and discussed, the invention herein is not limited to the particular disclosed embodiments or alternatives but encompasses the full breadth and scope of the invention including equivalents, and the invention is not limited except as set forth in and encompassed by the full breadth and scope of the claims herein.

What is claimed is:

1. A method for analyzing unauthorized intrusion into a computer network, the method comprising:

allowing access to an apparently vulnerable virtualized decoy operating system running on a hypervisor operating system hosted on a decoy network device;

using an introspection module comprising a virtual-machine-based rootkit module and its associated userland processes running on the hypervisor operating system to intercept a network attack on the virtualized operating system, wherein the network attack includes attack-identifying information; and

generating forensic data on the network attack from the attack-identifying information.

2. The method of claim 1, further comprising:

generating an attack signature from the forensic data; and

providing the attack signature to an intrusion prevention system configured to control access to a protected network using the attack signature to identify subsequent attacks.

3. The method of claim 2, further comprising controlling access to the protected network using the attack signature.

4. The method of claim 2, wherein the attack signature is automatically generated by the system without human intervention.

5. The method of claim 1, further comprising, before the allowing, opening a port on the virtualized decoy operating system through which the network attack is made.

6. The method of claim 1, further comprising:

allowing access to an additional virtualized decoy operating system running on the hypervisor operating system;

using the virtual-machine-based rootkit module to intercept an additional network attack on the additional virtualized operating system, wherein the additional network attack includes additional attack-identifying information; and

generating additional forensic data on the additional network attack from the additional attack-identifying information.

7. The method of claim 6, further comprising:

generating an additional attack signature from the additional forensic data; and

providing the additional attack signature to an intrusion prevention system configured to control access to a protected network using the attack-identifying information.

8. The method of claim 1, wherein the virtualized decoy operating system and the additional virtualized decoy operating system are different types of operating systems.

9. The method of claim 1, wherein said forensic data is generated based on an attack payload including keystrokes, ASCII, or binary files.

10. The method of claim 1, wherein the attack forensics are generated if an attacker is able to successfully gain access to the virtualized decoy operating system.

11. The method of claim 1, further comprising providing a report of said network attack to a network administrator.

12. The method of claim 1, further comprising storing the attack forensics in a database.

13. A system for analyzing unauthorized intrusion into a computer network, the system comprising:

a virtualized operating system module comprising:

a hypervisor operating system comprising:

at least one virtualized decoy operating system;

a virtual-machine-based rootkit configured to intercept a network attack on the virtualized operating system, wherein the network attack includes transmission of attack-identifying information; and

a processing module electrically coupled to the introspection module via a network interface communication channel, wherein the processing module comprises: a database configured to store forensic data on the network attack.

14. The system of claim 13, wherein the processing module further comprises an attack signature-generation engine configured to generate an attack signature from the forensic data on the network attack, wherein attack signatures may be generated on a timescale ranging from near-immediate to several minutes after initiation of an attack.

15. The system of claim 14, wherein the processing module further comprises a web-based visualization interface that facilitates configuration of the system and forensic analysis of captured attack information by administrators.

16. The system of claim 15, further comprising an intrusion prevention system electrically coupled to the signature-generation engine.

17. The system of claim 13, wherein the intrusion detection system is configured to prevent unauthorized intrusion into a protected computer network.

18. The system of claim 13, wherein the network interface communication channel is a private channel.

19. The system of claim 13, wherein the virtualized operating system module includes multiple virtualized decoy operating systems on the hypervisor operating system.

20. The system of claim 13, wherein the attack forensics are based on an attack payload including keystrokes or ASCII or binary files.

21. The system of claim 13, wherein the virtualized operating system module and the processing module are contained in memory on the same or separate computing devices that each includes a processor.

22. A computing device configured for analyzing unauthorized intrusion into a computer network, the device comprising:

a processor; and

memory coupled to the processor, wherein the memory comprises procedures for:

allowing access to a virtualized decoy operating system running on a hypervisor operating system hosted on a decoy network device;

using an introspection module running on the hypervisor operating system to intercept a network attack on the virtualized operating system, wherein the network attack includes attack-identifying information; and

generating forensic data on the network attack from the attack-identifying information.

23. The computing device of claim 22, further comprising a web-based visualization module comprising procedures for:

analyzing forensic data generated by the introspection module and the signature generation engine, as well as that stored in the processing module's relational databases; and

configuring the system, wherein an administrator can tune the system's behavior, including its pattern matching facilities, as well as automate the system's response to attack-identifying information captured by the introspection module and automate its response to forensic data generated by the signature-generation engine and any information stored on the processing module's relational databases.

* * * * *