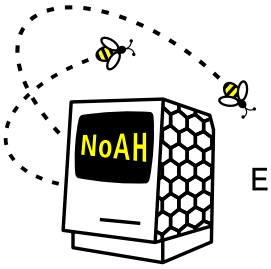
SIXTH FRAMEWORK PROGRAMME

Structuring the European Research Area Specific Programme

RESEARCH INFRASTRUCTURES ACTION



European Network of Affined Honeypots

Contract No. RIDS-011923

D0.1: Survey on the state-of-the-Art

Abstract:

The aim of the NoAH deliverable D0.1 is to summarise and to analyse the state–of–the–art related to the aims of the NoAH project. This document focuses on surveying existing honeypot architectures and security monitoring infrastructures, contrasting their relative advantages, and investigating their interoperability issues. The state–of–the–art is divided into the three categories *projects*, *tools*, and *scientific articles*. The projects are compared by introducing dimensions that reflect the advantages and disadvantages of each project. As a result, the technology gap is shown that NoAH is called to fulfill.

Contractual Date of Delivery	9/30/2005
Actual Date of Delivery	10/24/2005
Deliverable Security Class	Public
Editor	DFN-CERT
Contributors	FORTH, ETHZ, and VU

The NoAH Consortium consists of:

FORTH	Coordinator	Greece
VU	Principal Contractor	The Netherlands
TERENA	Principal Contractor	The Netherlands
FORTHnet	Principal Contractor	Greece
DFN-CERT	Principal Contractor	Germany
ETHZ	Principal Contractor	Switzerland
VTRIP	Principal Contractor	Greece
ALCATEL	Principal Contractor	France

Contents

Chapter 1

Introduction

The aim of the NoAH deliverable d0.1 is to summarise and to analyse the state–of–the–art related to the aims of the NoAH project:

- Design a state–of–the–art infrastructure of honeypots which will gather and correlate data on cyberattacks.
- Develop techniques for the automatic identification of attacks, and for the automatic generation of their signatures. Mechanisms to distribute these signatures to firewalls and other containment systems will also be investigated.
- Install and operate a pilot honeypot infrastructure to demonstrate the usefulness and effectiveness of distributed security monitoring systems.
- Collect information on attacks to examine trends, refine security models, and support Internet-related research efforts in general.

As a second aim, this summary and analysis is used to derive the technology gap that NoAH aims to fulfill.

The first part of the deliverable gives a survey on projects, tools and scientific articles that are relevant for the project. Projects and scientific articles are relevant if they follow similar aims in comparison with the NoAH project and if they seem to be important for the future work of the NoAH project. Tools are relevant if they may be integrated into the future architecture of the NoAH prototype. Because of the extend of the state–of–the–art this survey cannot be exhaustive and we focus on these contributions that are believed to be most relevant for NoAH.

1.1 Categories for the State–of–the–art Survey

The state–of–the–art is divided into tree categories:

- Projects

- Tools
- Scientific articles.

A **project** combines the application of distributed honeypots and similar sensors as well as deploying an infrastructure for the communication between these components with the analysis of the captured data. Captured data can include e.g. TCP/IP packets, data submitted to network services, or data which is captured by an host sensor (e.g. system calls). Considered are research projects as well as commercial projects.

The **tools** section summarises existing tools that are believed to be relevant for the NoAH project. A tool is understood as a piece of software which can be applied in an existing project. Categories for tools include

- High-interactive and low-interactive honeypots
- Intrusion detection
- Intrusion prevention
- Data capture
- Data analysis
- Tools that support an containment environment
- Tools that support an infrastructure for the transport of captured data

For the deployment of high-interaction honeypots it is crucial to prevent attacks that might originate from these honeypots. Therefore, it is important to detect intrusion attempts as well as compromises as early as possible. Tools that are able to detect intrusion attempts are classified as category *Intrusion detection*. To stop intrusion attempts from being successful, tools can be used that are classified as *Intrusion prevention*. For example these tools filter TCP/IP packets or drop TCP/IP connections that are established by an attacker. A different method to limit the risks is to prevent the attacker from doing previoulsly defined malicious actions on a compromised honeypot. These methods include the deployment of honeypots in a sand-boxed environment or the control of outgoing network traffic. These tools are classified as category *Containment environment*. Tools that provide a communication infrastructure (e.g. for the transport of alerts via the IDMEF-format) are classified as category *Communication infrastructure*.

This list of **scientific articles** includes approaches and techniques for data capture, deployment of honeypots, the related infrastructure, and data analysis that are believed to be most relevant for the NoAH project. There is no need that these techniques are already applied in an existing project or that a software program exists in which the technique is already applied.

Basically, the scientific articles are divided into three categories. The first category includes articles that aims at detecting spreading worms. Articles that cover approaches for the detection of malware by signatures or their generation fall under the second category. The third category is devoted to articles that are related to the deployment of honeypots.

1.2 Outline of the Deliverable

The rest of the deliverable d0.1 is organised as follows. Chapter 2, 3, and 4 summarises the state–of–the-art divided into the three categories. To evaluate the relevance of the projects, evaluation criteria and an evaluation matrix for the summarised projects is presented in chapter 5. Chapter 6 gives a summary of the results of the state–of-the–art survey and proposes which technology gap NoAH has to fulfill.

Chapter 2

Survey on Projects

2.1 DOMINO Overlay System

Introduction

The major aim of the DOMINO Overlay System (Distributed Overlay for Monitoring Internet Outbreaks) as described in [64] is to introduce a framework for the distributed storage and analysis of IDS data. The DOMINO architecture consists of different types of systems and provides a platform that is robust against denial of service attacks and network failures. In addition, the platform is designed to support a large variety of different IDS sensors and low-interaction honeypots by using a flexible data transport mechanism.

Architecture

The DOMINO architecture is arranged in a hierarchical structure and consists of the components *"Axis Overlay"*, *"Satellite Communities"*, and *"Terrestrial Contributors"* which have different tasks. Basically, the architecture combines both properties of peer-to-peer structures as well as hierarchical client server structures.

The axis nodes are the central components of the DOMINO architecture. All axis nodes are connected to neighboring axis nodes to share the data which has been locally captured. Thus, all axis nodes have a global view of the captured data and are able to find correlations in this data. Since the architecture uses a redundant set of axis nodes and is designed for a dynamically restructuring, it does not rely on the availability of a single server. The most important task of the axis nodes is to provide a component for decentralised storage and capture of IDS data ("Activity Database"). For data capture axis nodes provide "Active-Sinks" which emulate services in a block of IP addresses. Technically, they can be realised by using low-interaction honeypots (e.g. honeyd) or tools including the LaBrea tarpit. The data which is sent to the Active-Sinks is analysed and captured by network IDS (NIDS).

Satellite Communities are networks of satellite nodes. Their primary task is to capture data from additional networks and pass on this data to the axis nodes. They are directly connected to one of the axis nodes. Data from satellite nodes is considered to be less trustworthy than the data captured at the axis nodes. The Satellite Communities use the same sensor technologies which are used by axis nodes.

Terrestrial contributors are potentially very large sources for data and are located in arbitrary networks which need not to be directly participating in the DOMINO architecture. Therefore, the data is considered less trustworthy than any other data source in the DOMINO architecture. The terrestrial contributors are connected indirectly by access points to the axis nodes.

Honeypot and sensor architecture

The DOMINO architecture is open to a variety of sensor technologies. The sensors are deployed on all components in the DOMINO architecture and include low-interaction honeypots (e.g. honeyd), IDS sensors and firewalls. The data is exported using an extended version of the IDMEF message transport protocol.

Communication between honeypots and between honeypots and servers

Basically, an extended version of the IDMEF message transport format is used for the communication between the DOMINO components. Besides the native IDMEF message types ”*alert*” and ”*heartbeat*” new types are introduced. Periodically, ”*Summary*” messages are exchanged between peer axis nodes. They are produced by the aggregation of data that is received from the satellites by a pull-mechanism. Aggregation is e.g. used to create ”port”, ”source”, and ”cluster” summary messages. In addition to ”Summary” messages, ”*Alert*” messages are exchanged between axis nodes and between axis nodes and satellite nodes. In contrast to summaries, alert messages are spontaneously generated if an attack is detected by an IDS sensor. To reduce data and to globally prioritise the severity of alerts, these alerts are clustered among neighboring satellite nodes. Therefore, these nodes are structured in ad-hoc hierarchies. Other message types are used for registering new nodes in the DOMINO architecture and to issue queries to the database.

All communication between the axis nodes and the satellite nodes is encrypted. To prevent attackers from impersonating themselves as an axis node, all communication between these nodes is authenticated using public key cryptography.

Data capture

The captured data depends on the sensors that are deployed on the nodes. This data includes alerts which are generated by IDS and data from firewalls.

Containment strategy

Since only low-interaction honeypots are used there is no direct risk of being abused for attacks originating from the nodes in the DOMINO architecture.

Data analysis

Data which is captured on axis nodes as well as satellite nodes is aggregated. Aggregation on axis nodes is used to generate "*Summary messages*" which are exchanged between axis nodes. In addition, alerts are produced and clustered on the DOMINO nodes. The DOMINO axis nodes export an interface for queries concerning the captured data. These queries are exchanged between the axis nodes within "*Query messages*".

Efforts for deployment and supervision

The DOMINO architecture is designed to register new satellite and terrestrial nodes into the structured network. Therefore, new nodes can be easily integrated. Integration of new nodes and the restructuring of existing nodes is supported by *Topology messages*. Since only low-interaction honeypots are used there is no direct risk of being abused for attacks originating from the nodes.

2.2 cCSIRT.net Project

Introduction

The eCSIRT.net project in [5] is an European research project which is funded through the 5th EU Framework. Major aim is to deploy a widespread network of distributed IDS and to analyse the captured data. The IDS are mainly deployed by the European CERT community. However the eCSIRT.net initiative is open for participation by all European teams that have been shown to follow established best practices by joining the TI accreditation framework.

Anonymous statistical data which results from the IDS are presented on a public web-server. In addition, more detailed data is presented on a private part of the web-server which can only be accessed be the eCSIRT partners.

Architecture

The architecture consists of a distributed network of IDS sensors and a central server. All data which is captured by the IDS sensors is sent to this server and stored in a relational database.

Honeypot and sensor architecture

The IDS sensor is based on a knoppix linux distribution which is extended to include additional tools that satisfy the needs of the project. Since the complete distribution can be supplied by a boot-able ISO-image the system running the sensor can be very easily installed. In addition, the individual configuration of each sensor (e.g. IP address) is supplied by a floppy disk. On each eCSIRT sensor a Prelude network sensor and a honeypot daemon (honeyd) is deployed in order to capture data.

Communication between honeypots and between honeypots and servers

There is no direct communication between the IDS sensors. All communication is between the central eCSIRT server and the IDS sensors. For the transport of the captured data the native prelude architecture is used. The prelude IDS architecture consists of a set of sensors and a central manager. The sensors include daemons which capture data on the local system (e.g. system logging information) as well as daemons that capture and analyse all traffic within a network segment (Prelude-NIDS). All data captured by the sensors is sent to the central prelude manager using a secure SSL connection. For data transport the IDMEF format is chosen that is based on the XML language. For more details about the Prelude IDS we refer to the tools section.

The central server of the eCSIRT architecture offers a NTP service to all sensors. This service is crucial for the correlation in time of the captured data. For this service a secure implementation of the network time protocol (NTP) is used.

Data capture

The data is captured by the prelude network sensor (Prelude-NIDS). To be able to capture data sent to services, a honeypot daemon (honeyd) is installed that emulates specific services (e.g. web-server) and therefore, allows the attacker to establish TCP-connections to the sensors. Without this daemon the network sensors would not be able to capture any data contained in TCP-connections.

The Prelude-NIDS captures all attacks that match the corresponding Snort signature or that can be detected by a Plug-in of the Prelude-NIDS. Therefore, captured data is limited by the features of the Plug-ins and the availability of Snort signatures. Since only basic responses of a services are emulated, the system cannot be compromised by an attacker or worm. As a consequence of the limited emulation, any actions of an attacker or worm that follows the initial compromise cannot be captured. In addition, if the attack requires interaction of a service (e.g. if the attack requires a certain protocol state) the attack can only be detected if this interaction is emulated by the honeyd.

For data transport the IDMEF format is chosen which is based on the XML language.

Future sensor installations will also include the argus daemon in order to be able to capture network flows.

Containment strategy

Abuse of the system is not possible unless the Prelude-NIDS or the honyed are vulnerable to attacks. However, vulnerabilities in both products can be considered as unlikely. In additions, the sensors can be easily protected by a reverse firewall. Any connection originating from the sensor that does not concern the Prelude manager can be blocked.

Data analysis

All data is stored in a relational database on the central eCSIRT server. This data is used to create statistics which reflect the number and distribution of attacks.

Public available statistics include

1. The number of attacks which are recorded by each sensor
2. Daily statistics of attacks seen by the sensors
3. Monthly statistics of attacks seen by the sensors
4. Number of different kinds of attacks per attacker
5. Number of hosts that attacked more than one sensor

Efforts for deployment and supervision

Since the sensor is supplied as a boot-able ISO-image it can be very easily installed. Only very basic manual customisation is necessary to deploy the sensor. Because services on the IDS sensor are only emulated the effort for supervision is low.

2.3 LEURRE.COM Project

Introduction

The LEURRE.COM project in [10] is an international project that operates a broad network of honeypots covering more than 20 countries and the 4 continents. Primary aim of the project is to gather data that allows to better understand the current malicious activity. Therefore, the aim of the project and the technical realisation is very similar in comparison with the eCSIRT project. As a consequence, the project does not focus on the in-detail analysis of attacks which is e.g. in contrast to the honeynet project. Anonymous statistical data is presented on a public web-server.

Architecture

The architecture consists of a distributed network of low-interaction honeypots and a central server. All data which is captured by the honeypots is sent to this server and stored in a central relational database.

Honeypot and sensor architecture

Each LEURRE.COM sensor runs a modified version of the honeypot daemon (honeyd) that is designed to simulate services on a virtual Windows 98, Windows NT Server, and Red Hat 7.3 platform.

Communication between honeypots and between honeypots and servers

In analogy to the eCSIRT project there is no direct communication between the sensors. The central server connects to each sensor to download the captured traffic data. In addition, the integrity of the sensor is checked by the central server.

Data capture

Collected data include the full network packets that were captured on the honeypots and additional data including a guess for the operating system from which the attack originated. These additional data is obtained by the `b0f` or `Disco` tools for passive fingerprinting based on TCP/IP packet header information.

Containment strategy

Abuse of the system is not possible unless the honyed or programs that run on the sensor are vulnerable to attacks. However, vulnerabilities in both products can be considered as unlikely. In additions, access control is enforced and the integrity of the sensor is ensured.

Data analysis

As mentioned above, the primary aim of LEURRE.COM is to better understand the current malicious activity. Therefore, the project focuses on the identification of automated attack tools for whose the interaction given by low-interaction honeypots is sufficient. In [54] it is shown that the use of low-interaction honeypots is well-suited for that task.

Basically, a clustering algorithm is used to identify clusters of data that are mainly characterised by port sequences to which the malware connects to (see [53] for more details). Since each cluster is assumed to correspond to a specific attack tool (*root cause*) this approach allows to automatically identify these tools.

In addition, the captured data is used to create statistics which reflect the number and distribution of attacks. Public available statistics include

1. The spatial distribution and number of attacks divided into countries.
2. The top 5 probed port sequences and ports
3. The top 5 countries from which the attacks originate
4. The top 5 top level domains which the attacks originate

Efforts for deployment and supervision

Since the sensor is supplied as a boot-able ISO-image it can be very easily installed. Only very basic manual customisation is necessary to deploy the sensor. Because services on the sensor are only emulated the effort for supervision is low.

2.4 HoneyTank project

Introduction

The HoneyTank project as described in [61] aims at collecting and analysing large amounts of malicious traffic that is contained in TCP connections. Basically, the data is captured by an IDS (ASAX) that monitors all unused IP addresses in a network. The IDS emulates services on these addresses to capture data sent to services (e.g. web-server).

Architecture

Basically, the architecture consists of a single sensor on which an IDS (ASAX) is deployed. The IDS captures and analyses all traffic sent to unused IP addresses. The authors of HoneyTank propose to use mobile IPv4 (MIPv4) to redirect the traffic to the IDS sensor. To achieve this, the routers have to be configured to support the Home Agent functionality of mobile IPv4. To integrate a new unused IP address, the IDS sends a request for this IP address to the router for traffic redirection.

Honeypot and sensor architecture

All TCP/IP packets sent to the central sensor are captured by the libpcap library. This data is imported and analysed by the ASAX (Advanced Sequential Analyzer on Unix) IDS which is deployed on the sensor. The IDS allows the flexible declaration of rules that are applied to the captured data. Each rule declaration is written in the rule-based language "*RUSSEL*" and is used to analyse the captured packet and to apply actions that depend on the result of the prior analysis. The rule declaration allows the authors to emulate the basic behavior of the TCP protocol itself and the protocols HTTP and SMTP that are both based on TCP/IP protocol. The authors propose the performance of this approach to be advantageous because no

protocol state is maintained. Therefore, the number of parallel connections that can be emulated by the IDS is very high. However, the correctness of the emulation of the protocols is limited by the stateless behavior of the emulation.

Communication between honeypots and between honeypots and servers

All data is captured and analysed by a single ASAX sensor. The data is redirected to the sensor by using mobile IPv4. No other communication is present in the HoneyTank architecture.

Data capture

All data sent to the sensor is captured by the libpcap library. Therefore, the complete TCP/IP packets are stored in the native format of this library.

Containment strategy

Abuse of the system is not possible unless a program which runs on the sensor is vulnerable to attacks. Therefore, it is not necessary to contain attacks that may be originating from the sensor.

Data analysis

The data is used to generate statistical data of known types of attacks. An attack is known if a snort signature exists for detection.

Efforts for deployment and supervision

Integration of new unused IP addresses is done by sending a mobile IPv4 request to the router to redirect traffic. However since all network connections are processed by a single IDS, it is unclear how many IP addresses can be addressed by this approach. Since the services are emulated there is no risk of directly abusing the HoneyTank architecture for attacks.

2.5 Honeynet Project

Introduction

The honeynet project ([8]) is a non-profit organization that is devoted to the research concerning honeypots and their underlying architecture. Central aim of the project is the in-depth analysis of attacks and the capture of malware (e.g. IRC-Bots).

Associated to the project is the Honeynet Research Alliance, which is a research forum for independent organizations. Although the Alliance is independent

from the project the members share researching, developing and deploying honeypot technologies.

Architecture

The Honeynet project deploys an architecture that consists of a central gateway ("Honeywall") and the honeypot network. A central gateway separates the network in which the honeypots are deployed from the rest of the network. This gateway is called "*Honeywall*". The Honeywall performs access control of outbound connections from the honeypots and captures network data. Therefore, all traffic to or from the honeypots has to go through this gateway. Technically, the Honeywall is implemented as a filtering bridge. The systems that are deployed as honeypots are in a single network. All honeypots run different native operating systems. Thus, if an attacker is able to break into a honeypot the real machine is compromised. However, for an attacker it is very difficult to identify the system as a honeypot because all functionality that reveals the system as a honeypot is carefully hidden.

Honeypot and sensor architecture

All honeypots run the operating system on physical machines without emulation. Thus, the physical machine is intended to be compromised by an attacker.

As a disadvantage of this approach, the attacker gains full control over the compromised system after he has been able to compromise the account of the system administrator. As a consequence, the attacker is potentially able to detect all modifications of the system that are related to the honeypot functionality. Thus, these modifications have to been hidden very carefully. In addition, much effort has to be applied to prevent misuse of the compromised system.

Although the honeypot functionality cannot be hidden completely, this approach allows to efficiently disguise the function as a honeypot in comparison to other approaches. A crucial requirement is that all modifications of the original operating systems are carefully concealed. Alternative approaches use virtual machines as a platform for honeypots. However, most current software products (e.g. VMware) can be easily detected because of limitations of current hardware. These limitations can be overcome by the complete simulation of the hardware architecture (e.g. Bochs and Qemu). However, in general these approaches suffer from the poor performance which limits their deployment.

Deploying a high-interaction honeypot is advantageous, if monitoring the activities and the motives of an attacker is intended. This approach allows the attacker to download files and install tools on this system. E.g. after a successful compromise most of the attackers install a root-kit to hide their actions and to control the compromised system. In addition, IRC-Bots are a common tool to control a compromised system. In general, capture of this data is limited on systems that use low-interaction honeypots.

Communication between honeypots and between honeypots and servers

All honeypots are independent from each other. Thus, there is no communication between the honeypots related to the honeypot functionality. However, it is possible to simulate service traffic between the honeypots in order to convince the attacker that he has compromised a productive system. Data captured by the honeypots is sent to a central system that collects all data related to the compromise. As described in the section 'data capture' this communication is concealed from the attacker.

Data capture

Data related to the compromise is captured by the Honeywall as well as by the local system. The honeywall captures the complete TCP/IP packets sent to or from the honeypots.

Data on the local honeypot system is captured by the "Sebek" hidden kernel module. All data is directly sent to a central system outside the honeypot network. Using a kernel module allows to capture very detailed data related to the activity of the attacker. In addition, by using a kernel module, the capture of data can be effectively hidden from the attacker. Thus, all communication which is related to the captured data is disguised by the Sebek kernel module. Even if an attacker runs a packet sniffer on a compromised honeypot he will not be able to see these packets.

Containment strategy

The central system that prevents abuse of the honeypots is the Honeywall. The honeywall limits the rate of outbound connects from each honeypot. The choice of this number is crucial for the further analysis of the compromise. If it is too small the attacker will not be able to finish the compromise and may recognise the system as a honeypot. As a consequence, the information that can be gathered is incomplete. However, if the number is too high, the attacker might be able to use the honeypot to compromise other systems. In the white-paper "Know Your Enemy: GenII Honeynets" [24] it is proposed to use a limit of 15 outbound TCP connections a day.

In addition to the connection limit, a Network Intrusion Prevention System (NIPS) is deployed on the Honeywall. If the NIPS detects a known attack by a signature based approach, the outbound connection related to this attack can be terminated before the attack is finished or the payload of the packet can be modified. This NIPS is especially useful to prevent further attacks originating from Internet worms.

Data analysis

The analysis of the data follows multiple aims. A major aim is the manual forensic analysis of the data that is provided with the "*Honeynet Challenge*". In these challenges data which is captured on a honeypot is supplied to the security community for an in-depth analysis. The results are presented on the public web-server in order to share the knowledge about techniques and tools concerning the analysis of captured honeypot data.

An aim that is becoming more important is the tracking of bot-nets. Bot-nets are networks of IRC-Robots that connect to an IRC-channel on one or more IRC-Servers controlled by attackers. IRC-Robots are installed on compromised hosts and are used to control this host by commands which are distributed by the IRC-Channel the host has joined. Therefore, a Botnet allows the attacker to centrally control all IRC-Robots that are connected to the IRC-Channel. The Bot-nets are commonly used for distributed denial of service attacks and to relay spam-mails. A Bot-net can be efficiently tracked by using the captured IRC-Robot to gather informations about the Bot-net. The IRC-Robot can be detected if it connects to an IRC-Server. For a detailed description how to track Bot-nets we refer to [25].

Efforts for deployment and supervision

Aim of the honeynet project is to observe the actions of an attacker or malware in detail. For that reason, native vulnerable machines are deployed that offer the highest possible level of interaction and allow a forensic investigation of the modifications. As a drawback of this approach, the attacker has the full control over the compromised honeypot. As a consequence, he has the potential to attack other systems and to modify the compromised system. Although the risk of attacking the outside systems is basically limited by the honeywall, the risk of attacks cannot be eliminated completely. Therefore, much effort has to be spend into the manual supervision of the honeypots.

2.6 HoneyStat Project

Introduction

HoneyStat (see [34]) is a research project at the Georgia Institute of Technology. The major aims are to detect and analyse new attacks and zero-day worms and to apply statistical analysis to the worm behavior.

Architecture

The architecture is comprised of a central server (*Analysis node*) and distributed *HoneyStat nodes*. The HoneyStat nodes are the honeypot components that run multiple instances of emulated operating systems. If a compromise of an honeypot is detected, the captured data is sent to the Analysis node. The primary intention

of this node is to correlate honeypot activity, perform statistical analysis, and issue alerts.

Honeypot and sensor architecture

The HoneyStat nodes are the honeypot components running VMware GSX Server. This server allows to emulate multiple instances of operating systems on the same node. Each operating system has assigned 32 IP addresses. Therefore, a single HoneyStat node is able to support a large number of IP addresses. To address the problem of honeypot evasion, trivial patches are applied to the VM instances. The patches hide obvious indicators for VM instances (e.g. VMware registry keys). However, more complex tests for the existence of software emulation can not be fooled because the VMware architecture can not be hidden completely.

Evidence of a successful attack can be detected by tracking manipulations of the file system or violations of the memory management on the local system, and by observing disallowed network activity originating from that system. In [34] the term "*DiskEvent*" is used for a malicious manipulation of the file system, "*MemoryEvent*" for a violation of the memory management, and "*NetworkEvent*" for disallowed network activity. The native HoneyStat node as well as the emulated operating system have to be configured to detect these events. DiskEvents can be detected e.g. by applying tools that tracks the integrity of the emulated operating system (e.g. tripwire). These tools can be applied on the native operating system of the HoneyStat node. To detect MemoryEvents, tools like "stackguard" are suggested in [34] to be applied on emulated operating systems. Among other things these tools are able to detect buffer overflows in the stack segment which are assumed to be caused by an exploit for a vulnerable service.

Communication between honeypots and between honeypots and servers

There is no direct communication between the HoneyStat nodes. All communication is between the central Analysis node and the HoneyStat nodes. For the communication between the HoneyStat nodes and the Analysis node a secure channel is used. After data analysis the analysis node decides whether changes in the distribution of the deployed operation systems are necessary. These changes may be important to track worms or attacks that affect only a special type of operating system.

Data capture

The captured data if a compromise is detected includes:

- OS and Patch level for the compromised host
- Type of event (Memory, Disk, or Network)
- Data related to the event

  - DiskEvent: recent file changes
  - MemoryEvent: Stack trace, core file
  - NetworkEvent: network packet that caused the event

- file detailing recent network activity

Containment strategy

The HoneyStat nodes are configured to prevent the emulated operating systems from generating outgoing network traffic. Therefore, all network attacks originating from the HoneyStat nodes are prevented.

Data analysis

All events are evaluated on the central Analysis node. The major aim is to find correlations between captured events and network activity. In the HoneyStat framework logistic regression is applied to the events to identify these correlations. In this approach the network activity including noise related to port scans is considered.

Most automatic worms compromise systems by attacking a vulnerable service which listens on one or more TCP/UDP ports. This approach allows to find a correlation between these ports and the worm attacks. This correlation can be seen as a signature for a worm and therefore, the approach is able to identify worm outbreaks.

Efforts for deployment and supervision

The HoneyStat nodes are configured to prevent the emulated operating systems from generating outgoing network traffic. Therefore, all network attacks originating from the HoneyStat nodes are prevented.

Since the operating systems on the HoneyStat nodes are emulated, they can be easily deployed by using a copy of an operating system image. If a successful attack is detected the native operating system shuts down the emulated operating system that is compromised. After detection a fresh copy is used to replace the compromised image. Therefore, only very basic manual effort should be necessary to deploy the honeypots and to replace compromised instances.

2.7 Collapsar Project

Introduction

Collapsar (see [34]) is a project at the Purdue University. The project aims at deploying and managing a large number of coordinating high-interaction honeypots in different network domains. The proposed architecture called *Collapsar* allows

a centralised operation of the decentralised honeypots. A testbed of the Collapsar architecture is deployed at the Purdue University.

Architecture

The architecture is comprised of a central *Collapsar center*, traffic *redirectors*, and *front-ends* to the Collapsar center. The Collapsar center hosts a network of high-interaction honeypots, a central management station, and a central correlation engine. Key contribution of the Collapsar project is the redirector that allows to virtually deploy honeypots in arbitrary networks. A redirector is present in each production network that should be supervised. The basic function of the traffic redirector is to forward all traffic directed to unallocated IP addresses to the honeypots in the Collapsar center. As additional functionality, the traffic is filtered by this component because it can contain sensitive information.

Technically, the redirector is implemented as a virtual machine running an extended version of UML (see [21]). This approach is preferred because it does not require administrative access to a network router to redirect traffic. For example, an approach that uses features of the BGP global routing protocol to redirect traffic is proposed in [45]. To be able to intercept network connections, the redirector allocates all unused IP addresses in the supervised network. Thus, the router passes all traffic to the redirector, that is not targeted at a production system. After filtering this traffic, it is redirected to the Collapsar center. In [34] the authors show experimental results concerning the latency of ICMP packets that are caused by the redirection.

The front-ends are gateways to the Collapsar center. They perform dispatching on the received packets to the intended virtual honeypots in the Collapsar center. To avoid bottlenecks multiple front-ends can exist in the architecture. As an additional task, the front-ends control the traffic sent from or to a honeypot to prevent misuse that may originate from a honeypot.

The third component is the Collapsar center that basically consists of the honeypots, a correlation engine and a management station. The authors propose to use high-interaction honeypots that are based on VMware or UML virtual machines.

Honeypot and sensor architecture

The high-interaction honeypots in the Collapsar center are based on VMware or UML virtual machines. Later version of the project will also support additional machines including Xen, Virtual PC, and UMLinux.

Communication between honeypots and between honeypots and servers

There is no direct communication between the high-interaction honeypots. All captured data is sent from the honeypots to the correlation engine.

Data capture

Data is captured on the honeypots by the guest operation system. In detail, data is captured by the in-kernel logging module in VMware-based honeypots or the kernel-based snort variant *kernort* on UML-based honeypots. These approaches allow to record all intrusion related data including the attacker's keystrokes.

In addition, network data is captured by the tools *tcpdump* and *Snort* on the front-ends that are able to see all network traffic sent from or to the honeypots.

Containment strategy

Since all honeypots are based on virtual machines, misuse of the honeypots can be easily prevented by shutting-down these machines by the management station. However, details are omitted that concern the appropriate state at which the attack should be stopped.

Data analysis

All data is evaluated at the central Collapsar center that includes a correlation engine. This engine is able to detect port scan activity, distributed denial of service attacks, and worm propagations by analysing correlations between the captured data. In addition, IRC-Botnets are detected.

Efforts for deployment and supervision

As a consequence from deploying high-interaction honeypots, manual as well as automatic effort is needed to prevent attackers from attacking other systems.

Since the operating systems on the Collapsar honeypots are emulated, they can be easily deployed by using a copy of an operating system image. If a successful attack is detected the native operating system shuts down the emulated operating system which is compromised. After detection a fresh copy is used to replace the compromised image. Therefore, only very basic manual effort should be necessary to deploy the honeypots and to replace compromised instances.

2.8 Distributed Honeypot Project

Introduction

The Distributed Honeypot Project at [11] aims at deploying dispersed honeypots across the Internet and at sharing the results with the security community. The project is organised by Andrew Lamb. Basically, the documentation of the project focuses on the description of the deployment of different honeypot systems. These include virtual and physical systems which are used as honeypots. In contrast to other projects, the architecture is kept simple. In the minimum, all components can by deployed on the same physical system.

Architecture

Basically, the Distributed Honeypot Project focuses on the deployment of different honeypot systems. The project does not consider a comprehensive framework in which all honeypots are integrated. Instead, different architectures are proposed for each honeypot. All honeypots are based on high-interactive systems given by vulnerable physical machines and virtual machines. Each architecture is comprised of components that provide the functionality for access control, data capture, and the honeypot itself. All components can be deployed on a single physical system if a virtual machine is used as honeypot. In contrast, the use of a vulnerable physical system requires to spread the components to different physical systems.

Honeypot and sensor architecture

The project documentation includes the deployment of a native Redhat Server 6.2, HP-UX 11.00, and OpenBSD 3.0 and the use of a virtual honeypot based on Connectix Virtual PC 5.2. There is no framework or technique (e.g. data export) which is common to all deployed honeypots. Thus, the project does not rely on a special type of honeypot. An integration of all honeypots into a comprehensive framework is not considered.

In the case of the Redhat Server 6.2 honeypot, the access control is provided by a separate system running a Checkpoint FW-1 firewall. This firewall restricts outgoing traffic from the honeypot to remote ftp, smtp, domain-udp, and http service ports. All network traffic to the honeypot is allowed and logged by the Checkpoint FW-1. An additional system is used to capture TCP/IP packets. This system is also used to store the syslog messages and keystroke data which are logged on the honeypot. In addition, a Snort IDS is deployed on this system. To detect all modifications of the filesystem of the honeypot, the tool *tripwire* is used.

For deploying a virtual honeypot Microsoft's Connectix Virtual PC 5.2 software is proposed. In analogy to VMware, this software allows to run multiple instances of Microsoft Windows on the same physical system. Technically, the functionality and technology of this virtual machines is very similar in comparison to VMware. Both machines use the physical processor for executing programs. Thus, the processor is not emulated in both virtual machines. The Input and Output devices are handled via the host operating system. However, in contrast to VMware, only Microsoft Windows operating systems are supported by Microsoft's Connectix Virtual PC 5.2 software. Analogously to VMware, Connectix Virtual PC allows to keep track of all changes in the file system of the virtual machine. Therefore, this data can be used to detect an intrusion and for forensic investigations.

Communication between honeypots and between honeypots and servers

There is no communication between the honeypots.

Data capture

The type of the captured data depends on the type of the deployed honeypot. We refer to the section 'honeypot / sensor architecture' for a list of the captured data. A method for data fusion is not proposed in the project description.

Containment strategy

A firewall restricts outgoing traffic to remote ftp, smtp, domain-udp, and http service ports. If this traffic is detected by the Checkpoint FW-1, the supervisor is informed by an email which is sent by the Checkpoint FW-1.

Data analysis

All data captured on the honeypots is analysed manually. The authors do not propose any method for data analysis.

Efforts for deployment and supervision

The projects does not use a homogeneous honeypot architecture. Therefore, the efforts for deployment and supervision depends on the type of the honeypot. Since the honeypots are intended to be compromised, manual supervision is needed in all architectures to prevent the honeypot from attacking other systems.

2.9 Vigilante Project

Introduction

Microsoft's research project Vigilante (see [32]) aims at the detection and containment of Internet worms. In contrast to other projects this project does not rely on honeypots or net-flow analysis to detect worms. Instead, it is proposed to integrate the detection of attacks into the operating system of a large number of systems to detect malicious behavior during program execution. In addition, the authors propose a type of alert whose correctness can be verified by all systems.

Architecture

The Vigilante project uses a peer-to-peer network as the underlying architecture. This is in contrast to most of the other projects which rely on a central server. To effectively contain Internet worms the project relies on a very large number of nodes in the peer-to-peer network. Therefore, the authors do not use a limited set of honeypots to detect worms. Instead, they propose a general modification of the operating system to detect malicious behavior during program execution.

Honeypot and sensor architecture

The authors of the Vigilante project propose a general modification of the operating system of a very large number of systems to detect Internet worms. A large fraction of Internet worms compromise the victim systems by exploiting a vulnerability of a service which can be accessed from the Internet. For the attack, most attacks rely on submitting malicious content to this service. This content exploits the vulnerability by modifying the control flow of the service in a way that the attacker is able to execute arbitrary commands (e.g. by triggering a buffer overflow and overwriting a function pointer). The authors use the term "Arbitrary Execution Control (AES)" for this kind of vulnerability.

The key idea of the Vigilante approach is to specify malicious modifications of the execution control flow and to detect violations of this specification. This malicious modification is assumed to be caused by injecting data that is submitted by the attacker (e.g. included in an overlong HTTP Get-request to trigger a buffer overflow). Therefore, the data-flow of all user-supplied data is tracked. If user-supplied data or data that is derived from it modifies the control flow of the service in a malicious way (e.g. if user-supplied data overwrites a function pointer), an alert ("*Self-Certifying Alert*" - SCA) is triggered and distributed over the network.

The alert contains information about the user-supplied data and a list of all relevant events affecting the control flow of the vulnerable service which triggered the alert. Therefore, each system can use the alert to test if it is vulnerable to this attack. This is basically done by replaying the recorded events and user-supplied data in the alert in a sandbox-environment on the local system. If this leads to the same malicious state that triggered the alert on the remote system, the alert is verified.

Communication between honeypots and between honeypots and servers

The Vigilante project uses a structured peer-to-peer overlay network as the underlying architecture. Structured peer-to-peer overlay networks include self-organizing routing mechanisms and allow therefore to provide a platform for decentralised services, including fast content distribution. In the Vigilante project the peer-to-peer network is used to distribute alerts among the nodes.

In order to make the network resistant to denial of service attacks and to prevent worms from propagating over this network the network the authors propose some extensions of the original overlay network design in [31].

Data capture

The data is captured by the operating system of the systems. This data includes user-supplied data and events which lead to the malicious behavior of the control flow (Self-Certifying Alert).

Containment strategy

The attack is detected before it is able to take over the control over the local system. Therefore, the malicious code is prevented from being executed. However, only attacks that modify the control flow of a vulnerable attack in the considered way are detected. E.g. worms that propagate by attacking systems with a weak configuration or weak service passwords can not be detected by this approach.

Data analysis

There is no centralised analysis of the data. The data which has been captured during the attack is used to generate an alert. In addition this data can be used to verify the correctness of the alert by other systems.

Efforts for deployment and supervision

Key idea of the Vigilante project is to integrate detection of attacks into the operating system. Therefore, the project does not distinguish between honeypot systems and other systems and the supervision of the systems is not directly considered. All attacks that modify the control flow are detected and these attacks are prevented from being successful. However, other attacks that do not modify the control flow are still possible. Therefore, if these systems are deployed as honeypots, supervision is necessary to detect the attacks that are not detected by the modified operating system.

The approach to detect attacks requires a modification of the operating system. In addition, a rewriting of each binary whose control flow is tracked, is required by the current implementation.

Chapter 3

Survey on Tools

3.1 Honeywall

Categories

- Data capture
- Intrusion prevention
- Intrusion detection
- Containment environment
- Communication infrastructure

Basically, the *honeywall* provided by the honeynet project ([8]) is a linux distribution that includes a bridged firewall and additional tools for data capture, intrusion detection, intrusion prevention, and traffic control.

Description

The *honeywall* is a linux distribution based on fedora core 3 that allows to deploy a bridged firewall and includes tools for data capture, intrusion detection, intrusion prevention, and traffic control. Since a bridged firewall does not modify any IP packets, the functionality of the firewall is transparent to all systems in the network and therefore, it cannot be easily identified. Note, that most firewalls modify IP-packets during the IP-routing of these packets (e.g. the TTL IP header field). In addition to packet filtering, the honeywall can limit the packet rates per time interval that can be sent from the honeypots. This property can be used to limit the scan exposure from fast spreading worms.

For intrusion detection and alert generating the NIDS *snort* is integrated into the honeywall. Flow data is captured by argus and the honeywall allows to import data from Sebek 3.x clients. Snort's extension *snort_inline* can be used to automatically react to intrusion attempts by dropping or modifying IP packets.

For the data analysis, a web based user interface called *Walleye* is provided that can be accessed over SSL. This interface displays the captured data and allows to find correlation between this data.

3.2 honeyd

Categories

- low-interaction honeypot

The honeyd as described in [56] provides the functionality of a single low-interaction honeypot as well as a framework for a virtual network of these honeypots. Thus, a complete network of hosts running different operating systems can be simulated on a single system by honeyd. This simulation also includes the routing topology in the network. Therefore, the attacker believes to have access to a complete network.

The honeyd provides the functionality of a low-interaction honeypot running an arbitrary operating system. Network services on the honeypot are simulated by using external programs which simulate the basic behavior of the service. In addition, the traffic can be relayed to a remote host on which a native service is run.

Description

The architecture of honeyd consists of the following components:

Central Packet Dispatcher The Packet Dispatcher processes all incoming TCP/IP packets. First, it queries the configuration database if a specific honeypot configuration exists for the corresponding destination IP address. Then the Dispatcher hands the packet and the configuration to the appropriate protocol handler.

Protocol handlers Protocol handlers exist for the protocols TCP, UDP, and ICMP. The handlers for TCP and UDP allow to hand network traffic to external programs that run on the honeypot and simulate the basic behavior of services. For communication between honeyd and these programs the standard file descriptors 'stdin' and 'stdout' are used. Therefore, it is easy to integrate own programs into the framework. In addition, these handlers can redirect connections to remote systems on which native services run.

In the current honeyd distribution, scripts are supplied which simulates the behavior of virus back-doors (kuang and mydoom) and many network services. This list includes network services for HTTP (e.g. MS IIS), FTP, POP3, and SMTP.

Personality engine All outgoing traffic is processed by the Personality engine. This engine allows each honeypot in the framework to impersonate an arbitrary operating system. Thus, this functionality can be used to deceive an attacker and e.g. to make him believe that he is attacking a Cisco router.

The honeyd allows to simulate up to 65535 different honeypots on a single system. Performance tests have shown that a 1.1 GHz PC is able to simulate thousands of virtual honeypots. However, the performance of honeyd depends on the complexity and performance of the simulated services which run on each honeypot instance.

3.3 Mwcollect

Categories

- Data capture
- Low-interaction honeypot

mwcollect (see [12]) is a tool for data capture that is intended to capture autonomous spreading malware and that can be deployed on different operating systems. Basically, the mwcollect daemon emulates a vulnerable service and captures the data which is sent to this service in order to exploit an vulnerability. After capturing, the data is analysed and stored in a database.

Description

The tool mwcollect simulates the basic behavior of services or common back-doors of trojan software in such a way that malware treats the system as if it is vulnerable. Thus, only the behavior that is necessary to lure known malware is emulated.

Therefore, the tool can be classified as a low interaction honeypot that simulates services. However, in contrast to the honeyd, mwcollect uses directly the TCP/IP stack of the native operating system and focuses on the capture and analysis of malware.

Technically, mwcollect is based on a modularised architecture. Modules exist for the simulation of vulnerable services and back-doors, for parsing and analysing shell-code, for downloading files from HTTP and FTP servers, and for logging the results. The data flow between the modules is provided by three dispatchers that invoke a function of the appropriate module and supply the data to this module. For example, if shell-code has been captured the dispatcher for shell-code passes this data successively to all shell-code parsing modules until a module is able to successfully parse the shell-code. Therefore, additional modules can be easily added to the mwcollect framework.

List of modules:

Vulnerability modules Vulnerability modules simulate the basic behavior of services or back-doors and accept network connections. Modules are available for Windows RPC interface and for the back-door of the bagle worm.

Shell-code parsing modules A generic module (*scparse-genbot*) exists that parses shell-code using regular expressions. The module is able to detect code fragments that decode XOR-encoded data and to decode this data. In addition, code fragments are detected that invoke command line programs (*CreateProcess*). The common use of the CreateProcess function is to invoke commands that download programs using the protocols HTTP and FTP. The author of mwcollect proposes to use virtual machines to analyse shell-code in future versions.

Fetch modules These modules are able to download programs using the protocols HTTP and FTP. The URLs for downloading these programs are supplied by either the vulnerability modules or the shell-code parsing modules.

Submission modules Submission modules allow to store the captured malware as file in the filesystem ('submit-file') or into a PostgreSQL database.

Log modules The module 'log-irc' supplies log messages to an IRC-Chat network. This module allows to efficiently monitor multiple mwcollect daemons that report all to the same IRC-channel. This technique is similar to the technique used to monitor IRC-bot networks. An additional module enables logging over the syslog facility.

3.4 iDefense Multipot

Categories

- Data capture
- Low-interaction honeypot

Multipot (see [9]) is a tool for data capture which is intended to capture autonomous spreading malware. Basically, Multipot simulates vulnerable services and captures the data which is sent to this service in order to exploit an vulnerability. After capturing, the data is analysed and stored in a database.

Description

Multipot is a tool that offers similar functionality as mwcollect. Both tools simulate basic behavior of some services or back-doors to lure malware that connects to these services or back-doors. The level of interaction of both tools is similar. Only the behavior of the services or back-doors is simulated that is expected by malware to lure it.

In analogy to *mwcollect*, Multipot is modularised. Modules exist for:

- Kuang worm
- MyDoom worm
- Bagle worm
- Optix remote access trojan
- Sub7 remote access trojan
- Lsass vulnerable service
- Veritas Backup vulnerable service

In addition, modules exist for the download of files by FTP, TFTP, and HTTP. Captured malware is stored in a MS Access 97 database.

3.5 Prelude Hybrid IDS

Categories

- Data capture
- Intrusion detection
- Communication infrastructure

The Prelude Hybrid IDS in [15] provides a framework that consists of multiple sensors, a central system for data storage and analysis (Manager), and a library supporting the communication between sensors and the Manager. The sensors can be distributed among different networks and send alerts to a Prelude Manager. In addition, a web interface is provided which allows the presentation and correlation of IDS alerts. The software is licensed under the GNU General Public License version 2.

Description

The Prelude Hybrid IDS framework consists of the following elements:

Prelude Sensors

The Sensors include a host based Sensor (LML) as well as a network based Sensor (NIDS). The Prelude-NIDS is based on the library libpcap which captures tcp/ip packets in the local network.

Prelude-NIDS (Network Intrusion Detection)

The first step in analysing the captured packets by the Prelude-NIDS is to test the validity of the packets. This test is necessary because an attacker may have inserted malformed packets to disturb the IDS and to evade the detection of an attack. After this initial processing the packets are sent to plug-ins which perform a higher-level processing of the packets. A plug-in exists that processes signatures in the Snort-IDS format to detect attacks. In addition, TCP stream reassembly is done by an other plug-in. Protocol specific plug-ins process the HTTP, FTP, and TELNET protocol data and provide a normalisation of the captured data. The normalisation is required to cope with attacks that try to evade the detection by using an uncommon presentation of the data (e.g. using unicode-encoded data in URLs). Other plug-ins allow to detect portscans and patterns for Polymorphic Shell Code.

Prelude-LML (Log Monitoring Lackey)

The Prelude-LML is a host-based sensor that is able to monitor and analyse messages in the Unix syslog format. The export of syslog messages is supported by nearly all operating systems and network equipment. In addition, plug-ins exist for processing a variety of different log formats. An additional plug-in is able to apply signatures specified by regular expressions to the log messages. If a signature matches an alert is generated and sent to the Prelude Manager.

Prelude Manager

Basically, the Prelude Manager provides a centralised storage of the alerts generated by the distributed sensors. Therefore, the Manager allows the analysis and correlation of different alerts. The alerts are processed by plug-ins that store the data in a relational database, in XML format, or in plain ASCII text format. In addition, Managers can be used as relays for alerts. Therefore, the Prelude Managers can be hierarchically structured to provide a scalable framework for the storage and correlation of log messages.

Prelude Library

The Prelude Library provides the functionality for the authentication of sensors and managers in the Prelude Framework and communication between these systems. SSL connections are used for assuring the confidentiality of the data. To prevent the Manager from blocking, an asynchronous event queue is provided by the Library. The IDMEF transport format is used to sent alerts from the host as well as from network based sensors to managers.

3.6 Snort IDS and snort_inline

Categories

- Data capture
- Intrusion detection
- Intrusion prevention

Basically, the snort IDS in [19] monitors all packets which are captured in a network segment. The captured packets are analysed by applying a specified set of rules to the captured packets. If a rule matches, an alert is generated. The Snort IDS supports an extensive functionality for the specification of rules. In addition, Snort supports multiple formats for the presentation of alerts. The Snort IDS integrates an Intrusion Prevention System (IPS) to prevent ongoing attacks from being successful in real time.

Description

In the "*Network Intrusion Detection Mode*", Snort monitors and analyses all network traffic which is captured in a network segment. For data capture the *libpcap* library is used. The first step in processing the packets is performed by the packet decoder. In this step all protocol specific data is extracted from the packet (e.g. TCP ports and IP addresses). After this initial step, the packets are processed by Snort's *preprocessors*. These preprocessors allow to flexible extend the functionality to detect attacks for which no Snort rule can be specified. E.g. these attacks include portscans and denial of service attacks using fragmented IP packets. In addition, Preprocessors in Snort include a TCP stream reassembly module and modules that decode protocol specific data (e.g. HTTP Requests) and data of RPC-calls. Other Preprocessors are able to detect portscans, IP fragmentation attacks, and ARP spoofing attacks.

After the captured data passed the Preprocessors the data is processed by Snort's detection engine. The major aim of the detection engine is to detect attacks by analysing the captured data. For each attack a rule is specified which characterises this attack. These rules are specified with Snort's rule language. To detect attacks, the detection engine applies successively all rules to the captured data. If a rule can be successfully applied, the attack is believed to be detected which is related to this rule.

Each rule is comprised of a header and an options section. Basically, the header section allows to restrict the rule to a specific protocol, IP source and destination address, and TCP/UDP source and destination ports. The rule only applies to packets if they match these requirements. In addition, an action is specified in the rule header. The action is executed if the rule can be successfully applied to the packets. Actions include the generation of an alert, to log or ignore the packet, or to

activate additional dynamic rules. Rule options form the heart of Snort's intrusion detection engine. They allow to restrict the rule to packets which contain a specific payload or TCP/IP header data. The "*content*" option e.g. requires a sequence of bytes as argument. This option is successfully applied if this sequence is detected in the payload of a packet or TCP-stream. Additional options accept regular expressions as arguments (*pcre*), or allow to restrict the rule to specific data in the TCP/IP packet header.

Snort's output modules allow to present the data in a flexible way. Modules exist for logging alerts to a syslog facility, storing alerts in a relational database or an ASCII text file.

An interesting feature of Snort is the "*Inline Mode*" which enables Snort's Intrusion Prevention System facility (*snort_inline*). Snort_inline cooperates with the packet filter iptables and allows Snort to filter packets based on Snort's rule-set. Thus, all packets are first analysed by Snort's detection engine. All packets are dropped if a "*drop*" rule can be successfully applied to these packets. Therefore, snort_inline provides an elegant method to prevent known attacks from being successful. In addition, the payload of packets can be modified by using snort_inline.

3.7 Systrace

Categories

- Intrusion detection
- Intrusion prevention
- Containment environment

Systrace (see [20]) is a rule-based tool that is designed to detect and prevent intrusions on a local host. The tool is described in detail in [55] and is motivated by the large fraction of exploits that inject own code into the execution flow to gain control over the compromised system. The key idea of systrace is to restrict the allowed resources of programs to detect and prevent injected code that violates this restriction from being executed. The allowed resources are specified by rules that restrict the system calls that a program is allowed to invoke.

Description

Systrace is a rule-based tool that is designed to detect intrusions and prevent them from succeeding. The approach is based on the specification of resources that are allowed to be used by programs and network services on the local host. This specification is defined by a policy that is basically a list of allowed system calls. Thus, an intrusion is assumed if a program or network service violates this policy. System calls provide the basic functionality of the operating system. For example, programs use system calls to open, read, and write to files, to bind a network port,

to execute an external program, or to establish a connection to a remote host. All system calls are invoked from programs in the user space and are executed by the kernel of the operating system. E.g. the execve() call invokes an external program. Thus, if a program calls execve() the corresponding function is executed by the kernel. The results are handed to the calling program. The kernel provides a central interface for the invocation of system calls.

Monitoring and access control of system calls of a program has two major advantages. On the one hand this allows a fine grained control over the actions of a program and the resources it has access to. On the other hand, the central interface allows to track the system calls in an efficient way.

A large fraction of exploits try to change the execution control flow of vulnerable programs or network services to execute own program code which is inserted in the program flow by the exploit. After the exploit gained control over a vulnerable program it commonly tries to spawn an interactive shell. The common behavior of worms after the initial compromise is to download the actual worm code and to execute this code. Most of the exploits for typical program errors in C-programs rely on injecting machine code in the vulnerable program. Nearly all of this code uses system calls in order to execute the previously mentioned actions.

The key idea of systrace is that most programs do not need to invoke arbitrary system calls with arbitrary arguments. Therefore, these resources can be restricted to the necessary minimum. This allows to detect all exploits or worms that uses disallowed system calls for the initial compromise or after it. In addition, it is more difficult for an exploit to cause harm because its resources are limited. For example, if the exploit has no access to network resources it will fail if it tries to establish a network connection.

To detect and prevent intrusions Systrace monitors all system calls that are invoked by a program. The restricted system calls as well as their arguments are specified by a policy which is a list of rules. Each rule is comprised of the name of a system call, an allowed argument and an action. For example, the argument can be a filename or a pair of IP address and TCP/UDP port on which a network service is allowed to listen. The actions include to allow or deny the system call which corresponds to this rule.

3.8 VMware

Categories

- High-interaction honeypot
- Data capture

VMware ([23]) allows to deploy multiple instances of virtual honeypots on a single physical machine. A virtual honeypot is run in a virtual machine environment. Therefore, the virtual honeypot has no direct control over the hardware

resources. In addition, the virtualisation allows to hide the underlying operating system which operates the virtual honeypots. As an advantage, the underlying operating system can be used to capture data, to detect compromises and to control the actions of an attacker on the virtual honeypot.

Description

VMware allows to run multiple instances of virtual machines on the same physical machine. In contrast to a physical machine, a virtual machine is an (software) environment which provides virtual hardware resources (processor, memory, disk drives) to software which is run in this environment. Thus, the operating system of a virtual machine has no direct access to the physical hardware resources. Instead, the hardware resources are simulated by the software which runs the virtual machine (e.g. VMware). Therefore, multiple virtual machines can share the same physical hardware resources of the host system.

For performance reasons VMware mirrors parts of the physical hardware resources to the virtual machine. For example the native CPU and graphics hardware is directly used by the virtual machine. This is in contrast to different virtualisation software which simulates the complete hardware resources including the CPU. Although this approach is much more flexible the performance of the virtual machine is in general very low. However, this allows to completely hide the simulation. In contrast, the VMware approach does not allow to completely disguise the virtual machine simulation.

In VMware a native operating system is run in a virtual machine. Thus, after creation of a virtual machine by VMware, an operating system has to be manually installed on this machine. The current VMware Workstation 5 supports all current Microsoft Windows operating systems, most of the current Linux distributions and some Unix operating systems.

A virtual honeypot has some major advantages compared to the direct deployment of a honeypot on a physical system:

Control by guest operating system Unless there are any vulnerabilities in the VMware software, an attacker is not able to compromise the host system after a successful compromise of the honeypot. This is advantageous, because the attacker is not able to obtain the complete control over the physical machine. Thus, after a successful compromise any further attacks can be prevented by shutting down the virtual machine. As an additional advantage, the host system can be used to monitor the state of the virtual machine and to capture data.

Ease of deployment After the initial installation of the virtual honeypot, the virtual honeypot can be easily duplicated. Thus, it is easy to deploy multiple instances of virtual honeypots. In addition, after a successful compromise the honeypot can be easily reinstalled.

Intrusion detection and data capture The underlying host operating system has the complete control over the virtual honeypots. VMware allows to monitor and record all disk activity of the operating system of the virtual honeypot. Therefore, an intrusion can be detected by looking for unauthorised disk activity. In addition, this information can be used to track the activity of the attacker.

Preservation of state VMware allows to preserve the state of a virtual machine. Thus, the state of a virtual honeypot can be preserved which includes the capture of the complete memory and disk state. In addition, it is possible to revert repeatedly to the same state and to replay actions of an attacker.

3.9 QEMU and Bochs

Categories

- High-interaction honeypot
- Data capture

QEMU ([16]) and Bochs ([1]) are emulators for a hardware environment that includes multiple hardware platforms. In contrast to other virtualisation approaches, the complete hardware environment including the CPU is emulated.

Description

QEMU and Bochs are emulators for a specific hardware environment that allow to emulate multiple CPUs and offer a very similar functionality. Supported CPUs include Intel x86, PowerPC, ARM, Alpha, MIPS, and Sparc processors. Concerning the hardware environment, a memory management unit (MMU), Ethernet network interfaces, a SVGA graphics card, and PCI IDE interfaces are emulated. In contrast to other virtualisation approaches that map physical hardware resources to the virtual machine (e.g. Xen and VMware), the complete hardware environment including the CPU is emulated. Therefore, the machine code of the guest CPU has to be translated to the machine code that the host CPU can process. As a disadvantage, this translation results in a performance handicap compared to the aforementioned approaches. However, this technique allows to cope with hardware limitations concerning the Intel x86 CPU and offers more flexibility.

For performance reasons, the QEMU developers concentrate on the efficiency of the ”dynamic translator” that translates the machine code at runtime. To be efficient, the dynamic translator stores the code that is going to be executed in a translation cache. Therefore, code stored in this cache can be executed multiple times without being transled more than once. QEMU and Bochs allow to monitor the state of the guest operating system by providing basic support for debuggers.

In addition, both emulators can be used to track the file system activity of the emulated system.

3.10 Sebek kernel module

Categories

- Data capture

Sebek ([17]) is a kernel module which is designed to capture data on a compromised honeypot. The tool is available for Linux, Microsoft Windows, OpenBSD and Solaris platforms. Because this module is designed to be deployed on a compromised honeypot, special efforts are required to disguise the existence of this tool. For that reason, Sebek is implemented as a kernel module that allows to efficiently disguise its existence and to capture all relevant data that is related to a compromise. As an additional advantage, data from services that use encryption (e.g. ssh) is captured as plain text.

Description

Key feature of Sebek is to intercept all read() system calls to capture data which is associated to this system call. The read() system call is used by programs to read data from files and devices including network cards and the keyboard.

If a program invokes a system call, the kernel executes code in the kernel space which provides the functionality related to this call. Internally, the kernel maintains a table in which a function pointer for each system call is recorded. If a system call is invoked by a program the kernel executes the kernel-code to which the corresponding pointer directs.

To intercept read() system calls, Sebek replaces the function pointer of this system call with a pointer to an own code. Thus, if a program calls read(), Sebek's version of this system call is invoked. Basically, this Sebek's read() is a wrapper for the original read() system call. This enables Sebek to capture all data that is associated with this system call without changing the system call's behavior from the point of view of programs calling read().

Because this module is designed to capture data on a compromised honeypot, special efforts are required to disguise the existence of this tool. To disguise its existence, Sebek modifies the linked list of kernel modules. This technique is also used by kernel root-kits like adore for the same reason. However, the same techniques that can be applied to detect kernel root-kits are also valid to detect Sebek. Therefore, this tool cannot be hidden completely. But, it is not easy for an attacker to detect the existence of this tool.

All captured data is sent to a remote host using a covert channel to hide this communication. In addition, this traffic cannot be seen by any other honeypot on which Sebek is deployed. Technically, the tool uses its own implementation of the

raw Socket interface. All packets are sent directly to the device driver. Therefore, there is no easy ability for an attacker to block the packets with a packet filter (e.g. IPTABLES) or to detect their existence using a packet sniffer (e.g. tcpdump).

3.11 Bro IDS

Categories

- Data capture
- Intrusion detection
- Intrusion prevention

The Bro Intrusion Detection System ([2]) is an open source implementation of a network intrusion detection system that is available under a BSD-like licence of the Lawrence Berkeley National Laboratory at the University of California. The functionality can be compared with the Snort NIDS.

Description

In analogy to Snort, Bro is a network based IDS that analyses traffic, captured by a network sensor.

The traffic is captured by the libpcap library and passed to the event engine. Basically, the event engine processes all raw TCP/IP packets and prevents the typical IDS evasion techniques from succeeding that are based on the behavior of the TCP/IP protocol (e.g. insertion of bogus TCP/IP packets). Therefore, all malformed packets or packets that are believed not to be received by the addressed host are discarded. In addition, the event engine keeps track of all connections and generates events that are related to these connections.

All events are analysed by the Policy script interpreter. For the analysis of the events, Bro provides its own language that is dedicated to the event processing. Analyzer modules based on this language exist for the protocols FTP, HTTP, DNS, SSL (handshake), Ident, and RPC. In addition, a module for the signature based intrusion detection exists. Snort signatures can be transformed to be accepted by Bro IDS using the Python script *snort2bro*. In contrast to Snort's signature engine, Bro's signature engine allows to consider context information. For example, the result of a signature can depend on other signatures or boolean functions can be specified.

3.12 NFDUMP and NFSen

Categories

- Data capture

- Data analysis

Description

NFDUMP ([13, 14]) includes tools for the import and analysis of NetFlow data. NetFlow is a format defined by Cisco designed to provide statistics on packets flowing through a network. Basically, these statistics are comprised of data sets where each set corresponds to a TCP/IP connection and contains data such as IP addresses, TCP/UDP port numbers and the amount of transfered data related to this connection.

The NetFlow capture daemon `nfcapd` is able to import the NetFlow formats in version v5 and v7, e.g. exported by a router. For efficiency, imported data is stored in a *Round Robin Database*. This database is comprised of files in which each file contains the NetFlow data captured in a previously defined time slice.

The NetFlow data can be analysed by the graphical web-based front-end NFSen. This program is able to display the data captured by the NFDUMP tools. To narrow the flows down for a specific application, *Profiles* can be created which specify a view on the NetFlow data. For example, by creating a profile, the displayed data can be limited to flows originating from a specific network or directed to a specific TCP port number.

3.13 SIRIOS

Categories

- Data analysis

Description

SIRIOS (see [18]) is an extension of the trouble ticket system OTRS (Open source Ticket Request System), that provides a role-based system which allows to schedule tasks between different user groups. The extension is designed to satisfy the needs of Computer Emergency Response Teams (CERTs). In detail, SIRIOS allows to disseminate alerts in the IODEF and EISPP formats about computer security incidents that include the detection of a new vulnerability, the spreading of novel worms and the tracking of IRC-Botnets. Future extensions for the analysis and aggregation of data that are captured by darknets and honeypots are planned.

Chapter 4

Survey on Scientific Articles

4.1 Worm and Malicious Software Detection

4.1.1 Monitoring and Early Warning for Internet Worms

Zou et al. introduce in [67] an approach to early detect worms based on an epidemic model that is related to the spreading of infectious diseases. The authors use this model to estimate quantitative parameters that characterise the spreading of a worm.

Summary

Zou et al. introduce in [67] an approach that aims at the early detection of scanning worms. Therefore, they use a mathematical model often referred to as *SI-model*. This model assumes that the vulnerable hosts are homogeneous distributed into the Internet and that compromised hosts are scanning for vulnerable hosts by connecting to random IP addresses. In addition, the scan rate is assumed to be constant. The SI-model is fully specified by the scan rate at which the vulnerable hosts are contacted and the overall number of vulnerable systems. Once these parameters are known, the number of compromised hosts at any point in time can be computed.

In this contribution, Zou et al. use the SI-model to detect a spreading worm and to estimate the two parameters of the model. Therefore, they premise that the SI-model can predict the behavior of a scanning worm and that this assumption is in particular valid for the early stage of the spreading of the worm. The key idea of their approach is to test if the results from the SI-model are in compliance with the observed number of compromised hosts. If such a compliance is detected a spreading worm is assumed. A crucial precondition is, that this compliance can be detected in the early stage of a spreading worm.

Technically, the SI-model predicts the number of compromised hosts as a function of time. Note, that this function can be estimated from the observed scan activity or from the number of compromised honeypots. A compliance with the model is given if this function is correlated with the observed number of compromised

hosts as a function of time. In addition, the SI-model can be used to predict the future spreading of the worm.

In the contribution, Zou et al. simulate the spreading of the code-red worm. In their simulation they consider additional noise given by port scans that are not related to the worm activity. To detect worm scans they assume a sensor network that covers 2^{20} IP addresses and estimate the parameters of the SI-model by using a Kalman filter. As a result the authors demonstrate, that their approach reliably detects the worm at a stage where between 1% and 2% of the overall number of vulnerable hosts are compromised.

4.1.2 A Behavioral Approach to Worm Detection

The approach for worm detection proposed in [37] presumes that a network can be constructed that represents all communication flows between hosts in a network. This network is similar to a graph, but in contrast to a graph, a network allows two nodes to be connected by multiple links. Given this network, a worm is detected by searching the network for typical patterns related to the behavior of a spreading worm. If such a pattern is detected, it can be used to produce a signature. Since these signatures are related to the behavior of a worm they are denoted as "*behavioral signature*".

Summary

The authors use a network-theoretical approach to detect the spreading of worms. They assume that all communication flows in a network are known and that a network representing these flows can be constructed. The network-theoretical model introduced in [37] consists of an "*Abstract Communication Network*" and a "*Worm Propagation Network*".

In analogy to graphs, the Abstract Communication Network consists of nodes and directed links between two nodes. Nodes correspond to hosts and servers. Links correspond to a communication flow between two hosts or servers. To consider multiple network flows between two hosts, multiple links can exist between these nodes. Each link is related to a network flow. For example, two independent links can exist between the same two nodes:

```
- IMP request is sent from 'a' to 'b'
- TCP connection from host 'a' to host 'b' port 80
```

To model the behavior of a worm, a set of predicates is specified. For example, a predicate can be "*'a' sends 'b' an UDP packet to port 139*". In this contribution, predicates correspond to the characteristics of how a worm spreads. To give an example, the authors propose in [37] the following predicate related to the behavior of the Blaster worm:

```
P_Blaster(a,b) =
 -> 'a' connects to TCP/135 on 'b' and sends an exploit
   <- 'b' does not close its side of the connection
 -> 'b' sends packet to UDP/69 (tftp service) on 'a'
 -> 'a' replies with packet to UDP/69 on 'a'
```

The application of the predicates results in a transformed network which is denoted as "*Worm Propagation Network*" by the authors. The nodes of the Worm Propagation Network are the same nodes that are contained in the Abstract Communication Network. For the transformation, the set of predicates is applied to all links in the communication network. Two nodes in the Worm Propagation Network are connected if and only if they are connected in the Abstract Communication Network and if a predicate is true for the links between the nodes in this network. For example, nodes 'a' and 'b' are connected in the Worm Propagation Network if `P_Blaster(a,b)` is true for the links between host 'a' and 'host 'b' in the Abstract Communication Network. Because the predicates are related to the characteristics of a worm, all nodes in the Worm Propagation Network can be considered to be compromised systems. Two nodes are connected by a link if and only if the one system has directly compromised the other system. In addition, the predicates define a "*Descendant Relation*" between compromised systems that describes the order in which the systems have compromised each other.

In this contribution the authors propose two types of signatures which are based on the specification of predicates. Since the predicates are related to the behavior of worms the signatures are denoted as "*Behavioral Signatures*". The first type denoted as "*base signature*" is related to the view of a victim that is compromised by a worm. Victims first receive an exploit that is sent to a specific port that is open on the victim. After the compromise the victim begins to send exploits to other systems on the same destination port on which they received the exploit. This behavior can be specified by the combination of two predicates.

The second type (*inductive signature*) is related to the exponential growth of compromised systems during the spreading of a worm. It should be noted, that this is also assumed in [67]. As mentioned above, a Worm Propagation Network can be constructed from the Abstract Communication Network and the set of predicates. If this network shows an exponential growth over time the spreading of a worm is assumed.

4.1.3 Experiences using Minos as A Tool for Capturing and Analyzing Novel Worms for Unknown Vulnerabilities

Minos is a hardware architecture described in [33] that aims at detecting control flow attacks on honeypots. Basically the architecture is able to detect the exploit of buffer overflows and similar vulnerabilities where the flow of code execution is

manipulated by an attacker. Therefore, the Minos project share the same key idea with Microsoft's *Vigilante* project.

Summary

Most exploits for buffer overflows and similar vulnerabilities manipulate directly or indirectly the instruction pointer of the CPU to execute own code. In detail, many exploits overwrite the return address of the current activation record in the stack segment or a function pointer in the heap segment and replace it by a pointer that directs to injected code. Thus, the execution flow of the vulnerable service or program is redirected to the code of the attacker. The key idea of the Minos approach is to tag all data that is influenced by a user and to prevent all ways the data can manipulate the control flow. For example, Minos assumes an attack if tagged data is used as function pointer or return address.

The approach is very similar to Microsoft's *Vigilante* project that also aims at detecting manipulations of the control flow. In contrast to Vigilante, the authors propose Minos as a hardware based approach. A prototype is implemented by using Bochs to simulate a Minos hardware platform.

4.1.4 Accurate Buffer Overflow Detection via Abstract Payload Execution

The authors propose in [60] an approach that detects attacks in network traffic. The approach rely on the common proceeding of exploits for vulnerabilities like buffer overflows that redirect the execution flow to injected machine code.

Summary

Key idea of the approach introduced by Thomas Toth and Christopher Kruegel is that many exploits rely on the injection of machine code to which the execution control flow is directed during the attack. Since sequences of machine code are abnormal in most legitimate network traffic sent to services, the authors propose to detect attacks by looking for these sequences in network traffic.

The detection of machine code sequences is based on the interpretation of this code. In detail, the code is virtually executed to find the *maximum executable length (MEL)*. This is defined in [60] as the "*maximum of all execution lengths that are calculated by starting from every possible byte position in the sequence*". It should be noted, that jumps are considered during the computation of the MEL. To improve accuracy, the virtual execution of the code assures that all instructions in the sequence are valid. For example, if an invalid memory access is detected that would lead to the abruption of execution (e.g. a segmentation violation) the sequence is discarded.

4.1.5 Detecting Targeted Attacks Using Shadow Honeypots

The shadow honeypots introduced in [26] aim at detecting new exploits for vulnerable server of client programs by mirroring suspicious network traffic to these type of honeypots.

Summary

The shadow honeypots combine the advantages of network based anomaly detection and the use of honeypots. Most severe disadvantage of network based anomaly detection is the high rate of false-positives. This is because only anomalous traffic is detected and there is no accurate method on the network level to find out if this traffic is malicious or not. On the other hand, this decision is the primary strength of high-interaction honeypots. Therefore, the authors of [26] propose to mirror suspicious traffic to honeypots that has been previously detected by a network based anomaly detector. Because the network traffic is transparently mirrored to honeypots they are denoted as *shadow honeypots*.

Shadow honeypots allow to detect attacks against server programs as well as client programs. Since all traffic from a client can be mirrored to a shadow honeypot, the state of the client and the honeypot can be kept identical (the authors refer to this as *tightly coupled*). Therefore, shadow honeypots allow to detect exploits for client programs such as web-browsers that require user interaction. In the current implementation as described in [26], a modified memory management is used to detect buffer overflows resulting from successful attacks.

4.1.6 DIRA: Automatic Detection, Identification, and Repair of Control-Hijacking Attacks

The aim of the *DIRA* approach as introduced in [58] is to detect, identify and repair attacks which inject malicious code into the execution control flow of a vulnerable program. This type of attack is commonly used to overwrite critical control structures in programs in order to execute own malicious code.

Summary

C-programs contain a large number of memory structures that directly or indirectly control the execution control flow of the program. For example, if a function in a C-program is called, a memory structure (activation record) is written on the stack segment including a return address at which the execution control flow is continued after the function has been executed. However, because of insufficient bounds checking of a memory buffer on the stack, the return address can be overwritten by user-defined data. In this case, the attacker is able to replace the original address with an address, pointing to own code, which is directly executed after the function call (see e.g. [52] for further details). Other critical memory structures includes function pointers and data structures needed to execute the program (e.g. the global

offset table). Typical vulnerabilities allowing to hijack the control flow are given by the insecure usage of format strings or programming flaws affecting the memory management which e.g. lead to buffer overflows.

Technically, the DIRA approach is based on an extended version of the compiler Gnu GCC 3.3.3. As a consequence, each program or library that is intended to be protected has to be recompiled. The extended compiler inserts additional code into the program to detect, identify and repair control-hijacking attacks. Detection is done by inserting code assuring the integrity of critical memory structures. Thus, an attack is detected if the integrity is violated. Other inserted code keeps track of every memory update performed by the program. Basically, this is realised by logging the results of assignment statements (e.g. $x = y$) and by keeping track of library calls such as `memcly()` or `strcpy()`.

The information resulting from the memory update logging allow to identify the network packet containing the malicious data by which the memory structure was overwritten. In addition, this data can be used to repair the memory corruption caused by the attack. As an advantage, this approach does not necessarily require a complete restart of the program which might be very time-consuming. For that purpose, the program is restarted at a state in which the malicious data has not been received. Moreover, this technique allows to protect the vulnerable program from similar subsequent attacks.

Experimental results show, that the execution overhead in time is between 10% and 60%. The concrete value depends on the program that is protected by DIRA. In addition, the authors show that the approach is able to repair the consequences of an attack without restarting this program. This is advantageous, since a restart can be very time-consuming and repeatedly attacks can lead to a denial of service attack. However, this technique depends on the memory management and cannot be applied to every program.

4.2 Signatures for Detecting Worms and Malicious Software

4.2.1 Defending Against Internet Worms: A Signature-Based Approach

In [59] Tang et al. introduce an approach to automatically capture spreading worms and they introduce signatures (”*position–aware distribution signatures (PADS)*”) that are based on a statistical measure to detect polymorphic worms.

Summary

Tang et al. use two types of honeypots called *inbound* and *outbound* honeypots to capture worms. The inbound honeypots are high-interactive and are intended to be compromised. Therefore, all traffic that is originating from this type of honeypot can be considered as being malicious. For analysis and the generation of signatures

this traffic is sent to an outbound honeypot. Because of the two types of honeypots, the architecture is denoted as *double-honeypot* system.

A PADS signature is an ordered sequence of byte-frequency distributions with indices $1 < n \leq m$. Each byte-frequency distribution defines the likelihood of a byte-value to be present in a sequence of bytes that is contained in malicious traffic. For example, this sequence of bytes may be the shell-code of an exploit. Note, that this shell-code is observed very often in the malicious traffic by the outbound honeypots. Key idea of the approach is that although each byte-value of the exploit may vary due to the polymorphic nature, however, the statistical distribution of the byte-values is for all variants similar.

A signature of length m is applied to a sequence of bytes that has equal length and is extracted from the malicious traffic. In detail, a statistic similarity measure is computed in which the nth byte-frequency distribution is applied to the nth byte in the sequence. If this exceeds a threshold, the signature matches the payload. Because of the statistical nature of the signature, the authors propose that PADS signatures can be used to detect polymorphic worms. In addition, the authors propose a computation scheme to automatically generate signatures from payload samples.

For each position in the byte-sequence a different byte-frequency distribution is used. Because of this property, the signatures are denoted as *position-aware distribution signatures* (PADS). In analogy to this approach, a byte-frequency distribution is used in [62] to detect anomalous byte-sequences. However, in contrast to [59] the distribution is valid for every position in the sequence. This is in contrast to the approach of Tang et al., where the distributions depend on the position.

4.2.2 Anomalous Payload-based Worm Detection and Signature Generation

The authors introduce in [62] and [63] a sensor technology denoted as *PAYL sensor* that is able to detect unknown malicious payload. Since most worms show a self-replicating behavior, the authors assume that portions of incoming and outgoing malicious traffic are correlated that are sent by self-replicating worms. If a correlation in malicious worm traffic is detected, the resulting data can serve as signature data.

Summary

Key idea of the *PAYL sensor* in [62] and [63] is that the payload characteristic of legitimate traffic does not vary very much for each service and can be specified by a model. However, the characteristic of malicious traffic differs from the legitimate traffic. Thus, if traffic is detected which characteristic differs significantly from the model, this traffic is considered to be suspicious.

The model is based on the frequency count distribution of each byte. It is represented by a 256-element vector, in which the frequency of occurrence of each byte is stored. For example, for web-servers that provide only HTML based content,

the ASCII-Bytes corresponding to alphanumeric characters will dominate in this statistic. For each destination port and for each payload length different models are computed. The payload length is considered as relevant, because the frequency count distribution may vary with the length. For example, small payload length are likely given by the transmission of HTML pages and large payload length by binary files (e.g. images). To reduce data a clustering algorithm is applied that reduces the overall number of models by merging similar models.

All traffic that is sent to a service is compared with the data of the models that corresponds to the service and the payload data length. Technically, the authors propose to compute the simplified Mahalanobis distance between a 256-element vector given by a model and the actual 256-element vector that is computed from the payload data. Thus, this approach allows to compute a numerical distance between both vectors. If this distance exceeds a threshold, the payload is classified as malicious.

An additional aim of the approach is to find correlations in the payload of incoming and outgoing traffic. This aim is motivated by the self-replicating behavior of worms. In many cases the worm tries to compromise other systems by using the same exploit which has been used to compromise the victim machine. Therefore, it can be assumed that especially the exploit data is present in incoming as well as outgoing malicious traffic. Thus, correlations between this traffic may be an indication for a self-replicating worm. This correlation is detected by an approach that analyses the similarity of strings in the packet payloads. If the sensor detects malicious traffic to a destination port the packet payload is stored in a buffer. The payload of all outbound packets that are sent to the same destination port are tested for string similarities concerning the previously stored packet payload. The authors use different approaches for the string similarity:

String equality The egress packet payload has to be exactly the same as the ingress packet payload.

Longest common substring (LCS) The longest common substring which is present in the egress packet payload as well as the ingress packet payload is determined. If the LCS is long in comparison to the overall length of the packet payload a correlation between both packets is likely.

Longest common subsequence (LCSeq) This LCSeq is similar to the LCS. However the string do not need to be contiguous. This approach is supposed to be advantageous if small portions of the payload vary.

If a correlation between incoming and outbound packet payload is detected the results of the comparison of the packet payloads can be used to automatically produce a signature. This is possible because the correlation is detected by applying the aforementioned approaches for detecting string similarities. All approaches produce results that can be used as a signature.

4.2.3 Polygraph: Automatically Generating Signatures for Polymorphic Worms

This contribution in [48] describes an approach for the generation of signatures that is able to match the payload of polymorphic worms.

Summary

The contribution in [48] aims at producing signatures that match polymorphic worms. The major problem is that the payload of this type of worm varies. Therefore, signatures that are specified by a single contiguous non-variant sequence of bytes can in general not be applied to detect these worms. Instead, the authors propose signatures that are specified by sets of contiguous byte sequences (*Tokens*).

The architecture of the *Polygraph monitor* consists of a *Flow classifier*, a *Polygraph Signature Generator*, and a *Signature Evaluator*. The Flow classifier reassembles the payload of TCP connection and detects suspicious flows that are passed to the Signature Generator. In this contribution the authors concentrate on the Signature Generator and omit the design of the Flow classifier. The authors propose three types of signatures produced by the Polygraph Signature Generator:

Conjunction Signatures Conjunction Signatures are specified by an unordered set of Tokens. A byte sequence matches the signature if and only if it contains every Token in the signature.

Token-Subsequence Signatures Token-subsequence signatures are specified by an ordered set of tokens. A subsequence is any sequence of bytes that can be obtained by deleting zero or more symbols from a given byte sequence.

This type of signature matches if and only if every token in the signature is contained in the byte sequence. In contrast to conjunction signatures, the tokens must be contained in the same order as they are contained in the signature. Thus, the token-subsequence signature matches, if the sequence of tokens is a subsequence of the worm payload data. Token-subsequence signatures can be specified by regular expressions and therefore, they can be used to produce a signature for the Snort or Prelude IDS.

Bayes Signatures In analogy to the aforementioned types of signatures, Bayes signatures are also specified by a set of tokens. However, an exact match of every token is not necessary for the signature to match. Instead, a probabilistic decision is taken if the byte sequence is assumed to be malicious or not. This decision is based on a Bayes classifier which computes the likelihood that the signature tokens exist in a malicious and in a non-malicious sequence. The probability functions are computed by a supervised machine learning approach. Thus, the classifier has to be trained using known malicious and non-malicious byte sequences.

The experimental setup used by the authors to obtain quantitative results shows that the application of all proposed signatures leads to a lower rate of false positives compared to using the longest substring as signature. In addition, all signatures lead to similar or better results in comparison to using the best substring as signature. The authors propose that each signature has different advantages and disadvantages and recommend to consider the results of all types of signatures and to use the signature which seems to be superior for a specific application.

4.2.4 Automated Worm Fingerprinting

Singh et al. introduce in [57] an approach called *Earlybird* to detect novel worms and to automatically produce signatures for detected worms. Basically, their approach rely on the typical worm behavior and the assumption that at least one network packet produced by worms contain an invariant byte-sequence.

Summary

The approach in [57] rely on the following assumptions:

Content invariance It is assumed that the network packets produced by a worm contain a byte-sequence that is invariant.

Content prevalence Because of the self-replicating behavior of worms, the invariant byte-sequence is assumed to occur frequently in different captured network packets.

Address dispersion It is assumed that the number of infected hosts grows over time. Therefore, as a consequence, the number of IP addresses from which the packets containing the invariant byte-sequence originate will also grow.

The approach introduced by Singh et al. uses network-level data for detecting worms and producing signatures. All packets captured in the network are analysed by a single sensor. Basically, the sensor tries to detect packets containing an invariant payload or sequence of bytes that are captured frequently. Note, that this corresponds to the assumptions as described above.

To reduce computational complexity the authors focus on byte-sequences with a small fixed length. Technically, the occurrences of all byte-sequences of fixed length are stored in the *prevalence table* to be able to detect the most frequent sequences. To identify identical sequences the authors propose to use a variant of Rabin fingerprints that assigns each sequence a polynomial. Thus two equal sequences generate the same fingerprint. Once a frequently occuring byte-sequence has been identified, this sequence is used as a worm signature.

4.2.5 Polymorphic Worm Detection Using Structural Information of Executables

The authors propose in [46] an approach that detects attacks in network traffic. The approach rely on the common proceeding of exploits for vulnerabilities like buffer overflows that redirect the execution flow to injected machine code.

Summary

In analogy to [60], the approach introduced by Christopher Kruegel et al. is based on the extraction of byte-sequences from network traffic that contains executable machine code. However, in contrast to [60] the approach is extended to cope with polymorphic code that e.g. varies in each worm body. The approach is based on the assumption that it is much more difficult to modify the structure of the executable code than the machine code instructions itself. The structure of executable code is defined as a graph that is referred to as the *Control Flow Graph* in [46]. It is generated from the extracted machine code by static flow analysis and is mainly defined by the branches and jump instructions. To improve accuracy, the virtual execution of the code assures that the instructions are valid. This includes that all jump instructions jump to valid positions.

The signatures are based on the extracted Control Flow Graph. In other words, each signature corresponds with a specific graph. The algorithm to detect worms is similar in comparison to the Earlybird approach in [57]. In analogy to Earlybird a *prevalence table* is used to identify byte-sequences that occur most frequently in the network traffic. However, in contrast to Earlybird, the table is not constructed by counting the number of identical substrings that are seen in netflow traffic. Instead, the similarity of Control Flow Graphs is used that are extracted from the netflow traffic. Thus, the algorithm identifies worms by checking for frequently occurring byte-sequences of executable machine code from which the same Control Flow Graph can be constructed.

4.2.6 Autograph: Toward Automated, Distributed Worm Signature Detection

Kim et al. introduce in [42] an approach called *Autograph* to detect novel worms and to automatically produce signatures for detected worms. The approach is similar compared to the *Earlybird* that has been introduced in [57].

Summary

In analogy to *Earlybird* introduced in [57], *Autograph* tries to identify byte-sequences that are contained most frequently in the payload of network connection (*content prevalence*). Both approaches are motivated by the self-replicating behavior of worm and assume that because of this behavior invariant byte-sequences are in all

network connections originated by spreading worms (e.g. the exploit code) that are intended to compromise other systems.

In both approaches Rabin fingerprints are used to identify these common byte-sequences. However, in contrast to [57] the Autograph approach is only applied to those flows that are believed to be malicious. As an additional major difference, signatures are generated from the reassembled TCP-connection payload.

In the Autograph approach TCP streams are partitioned into variable-length content blocks. The length of these blocks depend on a *breakmark* that has to be predefined. A Rabin fingerprint is computed for all content blocks and used as a fingerprint to identify further occurrences of this content block in other connections.

4.2.7 An Architecture for Generating Semantics-Aware Signatures

Vinod Yegneswaran et al. introduce in [66] an approach for the generation of signatures that can be applied in network-based intrusion detection systems. Because their approach considers the characteristics of high-level protocols like HTTP they denote the resulting signatures as *semantic-aware signatures*.

Summary

Data is captured by a fully patched Windows 2000 server running on VMware and a virtual honeypot similar to honeyd. Because both systems are not advertised, the authors assume, that all captured traffic is malicious. The direct use of the Windows 2000 server to record SMB/NetBIOS connections is proposed to be disadvantageous, because in some cases connections were rejected before sufficient data concerning the attack were transfered.

The architecture introduced in [66] is denoted as *Neman Architecture* and comprised of the *Data Abstraction Component (ADS)* and the *Signature Generation Component (SGC)*. Basically, the ADS normalises and aggregates the packet traces which are captured by the honeypots in resulting connections and sessions. A session is defined by the sequence of related connections which have been established in respect to a certain protocol (e.g. HTTP request and response). The normalisation includes the reassembly of TCP connections and a normalisation which is specific to the corresponding protocol. For example, the decoding of URLs in HTTP requests is considered. Finally, normalised sessions are passed to the SGC in a XML-encoded format.

Signatures are generated for connections as well as sessions. First step in signature generation is to divide the connections into disjoint clusters. Each cluster contains a set of connections that are similar in respect to a given similarity metric. In [66] it is proposed to use a *Star Clustering* approach because it is independent on the order of the data and the approach does not require an a-priori knowledge about the number of clusters. As similarity metrics the authors propose to use the *cosine similarity* and the *hierarchical edit distance*. For each cluster, a signature

is produced by a machine learning approach that results in a finite state automaton matching all connections assigned to this cluster. In addition, signatures can be produced for sessions by generalisation of this approach. These signatures are also represented as finite state automatons. Since the signatures consider semantic characteristics of underlying protocols and can contain multiple connections, this type of signature is assumed to perform better than traditional signatures for NIDS as used in Snort or Prelude. However, the authors point out, that these signatures are suitable to be implemented for the Bro IDS.

4.2.8 Honeycomb - Creating Intrusion Detection Signatures Using Honeypots

Honeycomb is introduced by Kreibich and Crowcroft in [44] and aims at the automatic generation of signatures for spreading Internet worms

Summary

Honeycomb is an extension for the honeyd low-interaction honeypot that is designed to automatically generate signatures for spreading Internet worms. Currently, the tool is implemented as a plug-in for honeyd and can be downloaded from the project page in [7].

First step is to assign all incoming packets by protocol analysis to connections that are already stored in honeycomb. In this step the TCP stream is reassembled if the packet can be assigned to a TCP connection. If a new connection is established to the honeypot and payload is transmitted a new signature record is created.

In the next step honeycomb looks up all stored connections that have matching TCP/IP header elements in respect to the current connection to which the packet has been assigned to. Basically, the signature generation is done by the identification of common byte-sequences in the payload data of the current connection data and the data of the matching stored connections. Common byte-sequences are identified on the packet payload level as well as in the reassembled TCP-streams. The authors of honeycomb propose to use the LCS (Longest Common Subsequence) algorithm for that purpose. If an previously unknown byte-sequence is found, this sequence is proposed to be used as a signature.

4.3 Honeypots and Related Infrastructure

4.3.1 A Virtual Machine Introspection Based Architecture for Intrusion Detection

Garfinkel and Rosenblum propose in [38] a host-based IDS that detects compromises of an operating system running in a virtual machine. A prototype implementation *Livewire* based on a extended VMware version exists.

Summary

Garfinkel and Rosenblum propose in [38] an architecture for a host-based IDS. Basically, an IDS in the host system is deployed to supervise an operating system running in a virtual machine. The supervision is done by the virtual machine monitor (VMM) that provides an interface for the IDS to access and interpose status information of the guest operating system.

The authors propose that the benefits from this architecture are the *isolation*, *inspection*, and *interposition* of the guest operation system. Even if the attacker has full control over the guest operating system, access or modification of the host operating system and the IDS are not possible (isolation). However, the IDS has full control over the virtual machine, that enables the IDS to inspect the system state of the guest operating system and to interpose virtual machine operations. For example, this property can be used to detect malicious modifications of the instruction pointer of the guest operation system (e.g. caused by buffer overflow attacks).

The authors propose a list of policy modules that detect compromises. This list includes a policy for ensuring the consistency of the state of the guest operating system (e.g. caused by modified system binaries), a module to detect modifications of running processes, a signature detector, and a detector for the existence of raw sockets. In addition, modules are proposed that protect memory structures of the kernel from being modified (*Memory Access Enforcer*) and prevent NIC-cards entering in promiscuous mode (*NIC Access Enforcer*).

4.3.2 A Pointillist Approach for Comparing Honeypots

The aim of this contribution in [54] is to compare the observations given by low and high-interaction honeypots in a specific environment. As a major result, global statistics that result from the captured data of both types of honeypots are similar. Therefore, the deployment of high-interaction honeypots shows no major advantages for the aim of compiling statistics. However, the authors propose to use high-interaction honeypots are important for the correct configuration of low-interaction honeypots and to assure the quality and relevance of the results given by low-interaction honeypots.

Summary

This contribution is related to the french honeypot project at *Leurre.com* and aims at comparing statistical data resulting from low and high-interaction honeypots. For the comparison, three pairs of corresponding low and high-interaction honeypots are deployed. Each pair consists of a VMware image for high-interaction and a corresponding honeyd installation as low-interaction honeypot. To be able to compare the results, the honeyd is configured in such a way that all corresponding pairs offer the same services to the outside world and seem to run the same operating system. To assure the second requirement, the personality engine of honeyd has

been used. The pairs of honeypots run the operation systems Windows 98, Windows NT Server, and Red Hat Linux Server 7.3. Three contiguous IP Addresses are assigned to the low-interaction honeypots as well as to the high-interaction honeypots. The same order of the IP addresses in respect to the different operation systems has been used. In other words, the low-interaction honeypots have the IP addresses (x / Win98, $x+1$ / Win NT, and $x+2$ / Red Hat) and the high-interaction honeypots have the IP addresses (y / Win98, $y+1$ / Win NT, and $y+2$ / Red Hat), respectively.

In this contribution, Fabian Pouget and Thorsten Holz focus on three attack categories that are:

- Type I: Attacks that target only one honeypot
- Type II: Attacks that target two out of three honeypot
- Type III: Attacks that target all honeypots

The authors show, that overall number of attacks seen by the different types of honeypots is similar and that these results do not depend on the category of the attack. Therefore, they come to the conclusion that well-configured low-interaction honeypots are sufficient for compiling statistics of known attacks. To detect attacks, it is important that the honeypot offers the level of interaction that is necessary for the attack to proceed.

The authors of this contribution propose to use high-interaction honeypots to control the quality of the results given by the low-interaction honeypots. If there is a gap between the data resulting from a high-interaction honeypot compared with the data from the low-interaction honeypot the configuration of the low-interaction honeypot has to be updated. In addition, the data captured on the high-interaction honeypot can be used for this update. For example, this data can be used to create a honeyd script that simulates a specific vulnerable service.

4.3.3 Honeypot-based Forensics

Fabien Pouget and Marc Dacier propose in [53] an approach for the clustering of attacks seen by honeypots. The aim of the clustering is to identify attacks that are originated from the same attack tool.

Summary

Fabien Pouget and Marc Dacier propose an approach for the clustering of attacks seen by honeypots. The motivation of this clustering is to identify the cause of the attacks. Thus, the authors assume, that a specific tool causes similar data traces each time it is applied. Therefore, they propose that each cluster can be associated with a specific tool.

The clustering approach is based on a list of port sequences and a list of features. A port sequence is the sequence of TCP/UDP ports to which the malware

connects to during the attack. The sequence is ordered by the timely order of the connection attempts. In addition, the following features are used:

- The number of honeypots in the environment targeted by one attacking source.
- The number of packets sent by one attacking source to each honeypot.
- The overall number of packets sent by one attacking source.

For each port sequence the clusters are computed using a Association Rules data mining approach. By using this approach, the authors are able to identify clusters of data, which show e.g. an similar number of packets sent to each honeypots and a similar number of overall packets sent during the attack.

For the refinement of the clusters the Levenshtein distance concerning the attack payload is computed in each cluster. This distance is a measure for the similarity of two byte sequences. The cluster is split into sub-clusters if the variance of the Levenshtein distance in these sub-clusters is much lower than the variance in the original cluster. As an important result, the authors demonstrate that the clusters can be associated with an attacking tool.

4.3.4 A Hybrid Honeypot Architecture for Scalable Network Monitoring

The major aims of the contribution in [27] is to introduce an hybrid architecture that combines high scalability with a very accurate detection of new malware. Therefore, the authors propose an architecture which consists of cooperating low and high interactive honeypots. This cooperation is performed by a central control and command component. Thus, this architecture aims at combining the advantages of low and high-interaction honeypots while avoiding their drawbacks.

Summary

The major aims of the authors is to introduce an hybrid architecture that combines high scalability with a very accurate detection of new malware. A highly scalable architecture can be achieved by using low-interaction honeypots. Since this kind of honeypot cannot be compromised, it can be easily deployed without admitting any danger to remote sites. Therefore, the effort for containment and maintenance can be kept to a minimum. However, the low-interaction behavior limits the accuracy of capturing novel malware. This is because the simulation accuracy of services is in general limited. Low simulation accuracy may effect that the attack cannot be detected because it is disrupted in an uncompleted state. This drawback can be avoided by using high-interaction honeypots. This type of honeypot provides native services that can be compromised by an attacker. Therefore, all attacks can be performed and no problems arise because of a lack of interactivity of the service simulation. However as an disadvantage the scalability of a network of high interaction honeypots is limited.

In this paper the authors propose an architecture that combines the advantages of low and high interaction honeypots while avoiding their drawbacks. Key idea is that only new attacks are relayed to the high-interaction honeypots. The analysis of these attacks is of special interest and usually high-interaction is needed for the accurate analysis of these attacks. All known attacks are processed by low-interactive honeypots. Known attacks are in general interesting for compiling statistics, but should not pollute high-interaction honeypots.

The core problem of a hybrid architecture is the redirection of network traffic. The system has to decide whether to redirect the traffic to a high-interaction honeypot based on content of the traffic. The central problem of traffic redirection is that the decision to redirect the traffic cannot be made until the connection has been established and portions of the payload has been received. As a solution the authors propose to store initial session data in a cache and replay this session after the decision has been made. This decision is based on the first packet that contains a payload. If the payload can not be assigned to a known attack signature, the traffic is redirected to a high-interaction honeypot.

In detail, the proposed architecture consists of the following components:

- Lightweight (low-interaction) honeypots
- High-interaction honeypots
- Central Command and Control component

The major aim of the *Lightweight honeypots* is to accept connections from attackers, to filter the traffic payloads, and to redirect interesting traffic to high-interaction honeypots.

The Command and Control component coordinates connection redirection across sensors and provides a view of the global payload cache state. In addition, this component monitors the state of all high-interaction honeypots and provides a central platform for data capture and analysis.

4.3.5 A Survey on Virtualization Technologies

The contribution in [47] gives a survey on virtualization technologies. Virtualization is a technique that provides a transparent platform for software or operating systems that is simulated by software or hardware technologies.

Summary

By virtualization, an additional transparent layer is added on software or hardware platforms. This layer is used for the simulation or emulation of a software or hardware platform. For example, this layer can transparently simulate the complete hardware platform including the CPU. Therefore, the simulated machine that is provided by this layer can be denoted as a *virtual machine* since it behaves as the

original machine and it is transparent to the software or operating systems which run inside the virtual machine. In other words, software or operating systems that run in this virtual machine are not aware of the virtual machine and believes to directly run on the native platform.

Virtualization allows to run multiple independent operating systems simultaneously on the same hardware platform. In addition, a sandbox in which software or an operating system runs can be provided by this technology. Both properties are advantageous for the deployment of honeypots. This contribution summarises the existing technologies and approaches for virtualization.

The authors of this contribution distinguish between the following levels:

Virtualization at the Instruction Set Architecture Level

Virtualization at the instruction set architecture level includes the simulation of the physical CPU by a software solution. Thus, this technique allows to run e.g. a native Sun Solaris Sparc operating system on Intel x86 hardware platform. Usually, the major disadvantage of this technique is the low performance at which the guest operating system runs. This is because guest programs can not be directly executed on the physical CPU. Instead, each machine instruction of the guest operating system must be translated into a set of instructions that can be processed by the physical CPU. This translation and the additional simulation of the hardware environment causes this performance penalty. However, this technique enables a flexibility that allows to cope with hardware limitations. Even if the underlying hardware environment is not well suited for providing a virtual platform, the complete hardware simulation allows to cope with these limitations. In addition, as a major advantage for the deployment of honeypots, the virtualization can be completely hidden in respect to the guest operating system. However, the software that provides the virtual environment has the full control over the guest operating system and is able to monitor all actions or changes affecting the guest system. As an additional advantage, malware can not break out of the guest operating system and is not able to detect the simulation unless a vulnerability exists in the virtualization software.

This technology is implemented in Bochs and Qemu and provides a virtual x86 hardware platform. Bochs is open source program licensed under the GNU LGPL that can be deployed on most common platforms. As guest operating systems the emulation includes Linux, Windows 95, DOS, and Windows NT 4.

Virtualization at the Hardware Abstraction Layer

To cope with the problems of the aforementioned technique, the virtualization at the hardware abstraction layer executes the guest operating system directly on the physical processor and maps virtual hardware resources to the underlying hardware environment. Therefore, the performance penalty of the instruction translation is avoided. However, since most common hardware platforms including the x86 ar-

chitecture does not properly support virtualization, technologies have to be applied to cope with the hardware limitations.

Basically, the virtualization is provided by a Virtual Machine Monitor (VMM) that is implemented as an extra layer between the hardware and the virtual machines. This monitor provides virtual hardware resources to the virtual machines and maps these resources to the physical hardware. This is implemented by a transparent interface to hardware resources. Because of the transparency of this interface, the virtual machine believes to directly access the hardware resources. However, the I/O calls are intercepted by the VMM and processed by the hardware interface of the VMM.

As a major problem, the Intel x86 CPU is not well suited for virtualization. The CPU provides 4 rings which are associated with privilege levels. Most common operating systems rely on running in the most privileged ring 0. However, since the underlying host operating system runs in this ring, a guest operating system is not able to also run in this ring. To cope with this problems, different solutions have been introduced. VMware performs a code rewriting to trap each instruction in the guest operating system that requires privileged execution. After this instruction have been processed by the VMM the results are passed to the guest operating system. A slightly different approach is used by Xen. This approach requires modified sources of the guest operating system. Similar to VMware, the instruction that requires privileged execution are modified in such a way that they can be executed in a less privileged ring. Instead of the execution of the affected instruction, code is inserted that invokes a corresponding function in the VMM.

This technique is implemented in Microsoft's Virtual PC, VMware, Xen, and User-mode Linux.

Virtualization at the OS Level

In contrast to the aforementioned virtualization techniques, virtualization can be provided by the operating system itself. Basically, this technique is used for the partitioning of system and service resources at the level of the operating system. If multiple services run simultaneously on the same system that are independent on each other, each service can run in its own compartment. Thus, all services run on the same operating system instance, but they cannot access any files or see processes that are outside their compartment. Thus, if an attacker is able to compromise the system she or he is not able to access resources of other services. For example, the attacker cannot modify web-sites or access the database of the web-server after compromising the Email-server. This technique is implemented by the jail() system call of BSD variants.

Virtualization at the Programming Language Level

This technique is used to provide a virtual machine on the application level. A well known example of this technique is the Java Virtual Machine which can execute

Java Code. Key benefits are that these virtual machines are independent on the operating system and that they can provide a sandbox for untrustworthy software.

Virtualization at the Library Level

Key idea of this technique is that the functionality of an operating system that is used by applications mainly resides in libraries. Therefore, non native programs can be executed by providing an application binary interface (ABI) and an application programming interface (API) that is specific for this binary format. For example, WINE uses this technique to execute Windows executables on Linux platforms.

At the end of the survey the authors propose a novel technique for the virtualization of system resources which is called "*Featherweight Virtual Machine*". Their key idea is to duplicate data structures of the kernel in such a way that each virtual machine sees only their private copy of these data structures. Because this approach is similar to the name space of variables in programing languages that defines the scope of a variable, the authors call this technique "*name space virtualization*". The private data structures are related to the subsystems of the operating system kernel that provides memory management, process management, and the file system access. The Featherweight Virtual Machine is implemented by modifying the kernel interface for system calls. Thus, if an application invokes a system call, this call is intercepted and the modified system call interface assures that the virtual machine which runs the application works on its private kernel structures. The virtual machines are separated from each other by restricting the access to their private data structures. However, this technique does not allow a complete separation of the virtual machines until the virtual machines are able to access the kernel resources directly.

4.3.6 Mapping Internet Sensors with Probe Response Attacks

The contribution in [29] is devoted to the detection of Internet sensors that can be honeypots or distributed intrusion detection sensors. Their approach rely on the publication of statistics that include information about the number of scans in respect to the corresponding port.

Summary

The aim of this paper is to demonstrate the possibility to detect common network telescopes or honeypot networks that publish statistical results of the observed attacks. Their approach rely on the public statistical reports that include the number of scans for each TCP or UDP port. Any information about the source IP addresses is not necessary.

Basically, they use a divide and conquer algorithm to detect the locations of the sensors. Therefore, they split the total number of route-able IP addresses into n intervals. Key idea is to scan the IP addresses within each interval by sending packets to the ports that are less frequently scanned. These ports are scanned because the surveillance scan–activity can be most easily detected in the public reports.

If an increase of the number of attacks for these ports is observed for a specific interval, sensors are believed to exist in this interval. In addition the number of attacks is correlated to the number of sensors in this interval. If sensors are detected, the interval is recursively split into subintervals and scanned as described above. Thus, after some iterations, the IP addresses of the sensors can be determined.

The authors propose some countermeasures against their approach. These include the limiting of the rate at which the public reports can be downloaded, introducing a sampling rate for each sensor that limits the reported attacks, and to reduce the public information to the top scanned ports.

4.3.7 On the Design and Use of Internet Sinks for Network Abuse Monitoring

Yegneswaran et al. introduce in [65] a high-scalable architecture called *iSink* that provides the functionality of low-interaction honeypots.

Summary

The *iSink* (Internet Sink) architecture introduced in [65] aims at providing the functionality of a low-interaction honeypot focusing on high-scalability and high-performance. Technically, their architecture is based on the Click software project that provides packet routing functionality (see [3] for further information).

All captured network packets are passed to the IP Classifier that relay the packets to the appropriate responder module. Responder modules exist for ICMP Ping Requests, ARP, and for the TCP-based protocols HTTP and NetBIOS. For performance reasons, the responders do not keep any state of incoming connections. Experimental results demonstrate that the iSink architecture can cope with very high packet rates.

4.3.8 ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay

Aim of the ReVirt virtual machine introduced in [36] is to provide a virtual machine that allows to replay an arbitrary status of the guest operating system that runs inside the virtual machine.

Summary

A virtual machine is designed to provide a simulated hardware or software platform for a guest operating system. A benefit of this technique is that the status of the

virtual machine and therefore, the status of the guest operating system, can be observed and controlled by the host operating system. The key idea of the ReVirt approach is to keep track of all events that change the status of the virtual machine. By reversely replying all status changes, this approach allows to restore an arbitrary status of the guest operating system that runs inside the virtual machine. This technique is especially useful for the in-detail analysis of an intrusion.

Technically, ReVirt is based on the User Mode Linux (UMLinux) (see [22]). Basically, UMLinux allows to run a linux kernel in user space. Therefore, UMLinux can be used to realise a virtual machine (see [6] for details). For logging system status, ReVirt extends the UMLinux kernel to log all non-deterministic events, that affect the execution of the guest operating system inside the virtual machine. Deterministic events (e.g. a SIGSEGV signal) do not need to be logged because of the deterministic nature of these events. Note, that events initiated by virtual storage mediums like hard disks or CD-Roms fall in this category. Non-deterministic events include events that are initiated by input devices (e.g. keyboard, mouse, or network cards), hardware interrupts, and by the delivery of signals (e.g. SIGALARM). The exact timing of these events is logged which is is crucial to avoid race conditions. Otherwise, a deterministic result from replaying these events would not be possible. To reduce data from the ReVirt machines, a cooperative logging is implemented in the CoVirt Project in [4].

4.3.9 Detecting Honeypots and Other Suspicious Environments

Thorsten Holz and Frederic Raynal describe in [40] how common high-interaction honeypots can be detected that use virtual machines based on UMlinux or VMware. In addition, they discuss the influences of honeypot modifications on the timing behavior and demonstrate that `chroot()` and `jail()` isolations and debuggers can be detected. A detailed description how the SeBek kernel module can be detected and how it can be attacked is presented in [35].

4.3.10 Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention

Janakiraman et al. propose in [41] an approach for a distributed network intrusion detection and prevention system called *Indra (Intrusion Detection and Rapid Action)*. Indra is based on a peer–to–peer network that is used to distribute information about compromised systems and information about how to protect against new vulnerabilities. Thus, if a host in the Indra network is compromised and subsequently abused as a stepping-stone to attack other hosts, these hosts distribute a warning as soon as they detect the attack. In addition, the other hosts can be protected against attacks that originate from the identified compromised host.

Indra is motivated by the robust nature of peer–to–peer networks that are resistant against denial of service attacks that are directed against a single point of failure. Since the trust relationship between the hosts is very important to be re-

sistant against attacks that are originated from a host inside the Indra network, the authors propose to introduce a web of trust between all hosts. The detection of intrusions is done by Indra daemons that are running on each host in the network.

Chapter 5

Evaluation Criteria for Projects

In this chapter, criteria for the evaluation of all projects included in the state–of–the–art survey are introduced. These criteria are intended to stress the advantages and disadvantages of each project in respect to the aim and the technical realisation. The criteria cover the properties of the sensors as well as the overall architecture that are deployed by the project. In addition, criteria are introduced for the dimension of the deployment effort and the effort to prevent misuse of the architecture. Based on this evaluation, we conclude in the next chapter which solutions of projects are relevant for NoAH and we identify the technology gap that NoAH should fulfill. In the following, these evaluation criteria are introduced:

D1.1: Level of cooperation between honeypots

During the operation of the honeynet architecture the honeypots may share data, which is supplied by direct communication between the single honeypots. This data can include attack signatures, configuration specification, and containment information. This element provides an dimension of the direct cooperation between the honeypots. The range is from ’no (cooperation)’ to ’very high’. ’no cooperation’ means that there is no direct communication between the honeypots at all. If there is a minimal data exchange, the level of cooperation will be rated as ’low’. ’very high’ means that there is a very detailed and complex data exchange.

D1.2 Level of cooperation between honeypot and central server

During the operation of the honeynet architecture the honeypots may receive data from a central server or network system. This data can include attack signatures, configuration specification, and containment information. This element provides a dimension of the complexity of communication between the honeypots and the central server. The range is from ’no cooperation’ to ’very high’. ’no cooperation’ means that there is no direct cooperation between the honeypots and the server at all. If there is a minimal data exchange the level of cooperation will be rated as ’low’. ’Very high’ means that there is a very detailed data exchange.

D2: Security of network architecture

This item is used as a dimension for the security of the network architecture which is used for the communication between the honeypots and between the honeypots and a central server. Security means confidentiality, integrity, and authenticity of the data and the availability of the components of the infrastructure. 'Low' means that the components use basic tcp/ip connections without applying security measures (e.g. encryption, authenticity). If the communication is secured by encrypting all data and also the communication partners are authenticated (e.g. by using ssl-based connections) the value is set to 'medium' or higher. If the architecture is additionally secured against denial of service attacks or network failures (e.g. by deploying redundant systems) the value is 'high' or 'very high'. The concrete value depends on the complexity of the architecture.

D3: Scalability of network architecture

The value for the scalability of the network architecture is a dimension that concerns the integration of new sensors into the network. The value 'very low' means that scalability of the architecture is not addressed by the project and that the number of sensors in the framework is limited. If the number of sensors can be arbitrarily increased, but a huge manual effort is necessary to integrate sensors or to adapt the architecture to the increased number of sensors, the values 'low' or 'medium' are chosen. The value depends on the effort to integrate a new honeypot into the architecture. If the architecture is designed to address scalability the values 'high' or 'very high' are chosen. The concrete value depends on the effectiveness of the solution.

D4: Level of honeypot-interaction

Honeypots differ in the level of interaction they offer to attackers. The meanings of the values are:

No interaction There is no interaction between the attacker and the sensors. E.g. this value is selected if darknets are deployed.

Connections TCP/IP connections from attackers are accepted. However, attackers do not get any response except for the TCP handshake response.

Services (sl) Services (e.g. web-servers or Microsoft Windows RPC services) are roughly simulated. In addition, a vulnerability of these services may be simulated. However, the service is not intended to be compromised by attackers. If no state concerning the connections is kept, it is additionally categorised as "stateless (sl)". For example, the replied packets are produced by a stateless rule-based scheme.

Services (sf) In contrast to the previous item, all connections are simulated while keeping the connection state.

VM The operating system itself or vulnerable services are run in a virtual machine and can be compromised. However, the real operating system is hidden and can not be seen by the attacker. Interaction and access to software or hardware resources can be limited by the guest operating system.

Physical The attacker can compromise the native system and is able to gain the full control over the system. Interaction is only limited by external systems (e.g. firewalls).

D5: Level of deployment effort of the honeypots

The honeypot types vary in the effort that is necessary for deployment. Values range from 'low' to 'very high'. 'low' means that the honeypots can be deployed with minimal effort, e.g. by setting up a system with a boot CD. 'medium' is chosen if basic manual work has to be done for deployment. If complex manual work is necessary for deploying the honeypots, the value is set to 'high' or 'very high'.

D6: Level of supervision effort

The honeypot types vary in the efforts which are necessary for supervision. Values range from 'low' to 'very high'. Value 'low' means that the honeypots need only a very small effort for supervision (e.g. because the honeypot is not intended to be compromised). If misuse of the honeypot is believed to be limited or can be prevented automatically, 'medium' is chosen. If complex manual supervision is necessary to prevent attackers from abusing the systems, the value is 'high' or 'very high'.

D7: Level of complexity of captured data by network sensor

This item is used as a dimension for the complexity of data which is captured on the network level. Value **'NF data'** (netflow data) includes flow data concerning TCP and UDP connections. However the payload of the packets is omitted. If complete TCP, UDP, and other IP packets including header and payload are recorded the value is **'Packets'**. Value **'Alerts'** is chosen if attacks are detected by a NIDS and alerts generated by the NIDS are recorded.

D8: Level of complexity of captured local data

This item is used as a dimension for the complexity of data which is captured on the local honeypot. 'low' means that only data is captured which can be very easily accessed (e.g. packets from TCP/IP connections) and that no post-processing

is applied. If simple post-processing is applied or service data is captured (e.g. reassembled TCP/IP data streams) the value is 'low'. 'medium' is chosen if more complex post-processing is applied (e.g. if attack signatures are used to identify known attacks). If additional sophisticated approaches are used to capture data at system level (e.g. by modifying the kernel) the value is set to 'high' or 'very high'.

Project	D1.1	D1.2	D2	D3	D4	D5	D6	D7	D8
DOMINO	very high	high	very high	very high	Services (sf)	low	low	NF data	medium
eCSIRT	no	medium	medium	high	Connections	low	low	Alerts	medium
LEURRE	no	medium	medium	high	Services (sf)	low	low	Packets	medium
HoneyTank	no	medium	medium	high-medium	Services (sl)	low	low	Packets	low
Honeynet	no	low	low	low	Physical	very high	very high	Packets	very high
HoneyStat	no	medium	medium	low-medium	VM	medium	high	no	very high
Collapsar	no	medium	-	medium	VM	high	medium	Packets	very high
Lucidic	no	low	low	low	Physical/VM	very high	very high	Packets	high
Vigilante	high	no	high	medium	Physical	high	medium	no	high

Table 5.1: Evaluation matrix for projects

Chapter 6

Summary and Conclusion

This Deliverable gives a survey on 9 projects that are devoted to the deployment of honeypot-networks or networks of distributed intrusion detection systems (IDS). Due to the variety of different approaches, the aims of the projects differ very much.

The first major difference is whether the projects focus on the honeypot technology or on the architecture between the honeypots or IDS. The later applies to the DOMINO Overlay system that provides an open architecture for low-interaction honeypots and IDS. This architecture is comprised of a hierarchical structure of a large number of nodes that can host different types of low-interaction honeypots and IDS. Because of its structure it does not include single points of failure and is therefore resistant against denial of service attacks. In addition, low-interaction honeypots and IDS can easily be integrated into this architecture.

All other projects in chapter 2 concentrate on the honeypot technology. The eCSIRT and LEURRE.COM projects deploy broad networks of low-interaction honeypots. Primary objective is to get an overview over the current malicious activity. For the accuracy of the statistical data it is critical to deploy a broad distributed network of honeypots. A different low-interaction approach is proposed by the HoneyTank project. In contrast to the previous two projects, traffic is redirected by routers and relayed to an IDS that processes all connections. Target of this project is the detection of worms and automated attacks in ISP-scale networks. As demonstrated in Tab. 5.1, a major advantage of all three projects is the high scalability of the number of sensors. Thus, additional sensors can be easily integrated into the network without larger effort in maintenance. Although all projects include a single point of failure given by a central database server or IDS, these components could be replaced by a distributed component.

The other projects deploy high-interaction honeypots. Aims of these projects vary from the capture of malicious software to the awareness of new worms or

zero-day exploits. In addition, different solutions for high-interaction honeypots are proposed.

Microsoft's Vigilante project modifies the operating system in such a way that a common kind of attack can be detected. The proposed approach is especially advantageous in the identification of attacks at a very early stage. In detail, intrusions are detected as early as they are able to inject untrusted data into the execution control flow of the honeypot in a specific way (A similar approach is introduced by the Minos project in [33] and the contribution in [49]). Since the primary exploit vector is prevented, the attack is unable to succeed and it is very difficult for an attacker to evade this technique. As a consequence, even polymorphic worms are reliably detected which is in contrast to common network based approaches as shown in [43]. In addition, these approaches allow to produce signatures that detect attacks very reliably and are not prone to false warnings. Because of these advantages, the accuracy of this type of signature can be assumed to be superior to the common signatures of network based IDS. Moreover, host–based intrusion detection can be used to complement the application of widespread network based IDS or behavioral approaches as proposed in [37], [46] or [60] to generate more accurate signatures and to exclude false warnings. However, all types of signatures allow to use similar algorithms for an automatic generation (see [59, 62, 42, 57, 63, 44]).

A complementary approach is the use of sequences of system calls for intrusion detection and for producing signatures (see e.g. [39]). Fundamental idea is that most exploits use system calls to control the compromised system. Thus, malicious activity can be detected by tracking the system calls a process invokes for malicious activity (e.g. by *systrace*). In addition, this data can be used to produce complementary host–based signatures.

The honeynet and lucidic.net projects use vulnerable machines as honeypots. The honeynet project deploys a modified operating system kernel (*Sebek kernel module*) to capture data and to disguise the existence of the modifications. Therefore, as long as the attacker is not able to detect the modified kernel, the hosts cannot be identified as a honeypot. In addition, as shown in [28] all actions of the attacker can be recorded in detail and this approach allows to capture all malicious software installed by the attacker. As an additional advantage, IRC-bots can be tracked down by observing connections to IRC-servers. The honeystat project deploys honeypots based on a VMware virtual machine. The use of virtual machines decreases the efforts of honeypot deployment and improves the scalability of the number of honeypots. In addition, all status changes of the virtual machine can be tracked by the host operating system. This allows to identify intrusions and to replay the status of the virtual machine (see [36]). However, an attacker is able to identify the virtual machine (see [40]) as long as the emulation is not exact. The risk of detection may be reduced by using *Bochs* or *Qemu* which provide an exact emulation of the complete hardware platform.

A principal disadvantage of the deployment of high-interaction honeypots is the low scalability of the number of honeypots. This is due to the nature of this type of honeypots which allows the attacker to compromise the system. As a consequence, big effort has to be spent to limit the exposure of attacks that may originate from the compromised honeypots. An additional drawback is that a compromised honeypot must be reinstalled in order to observe subsequent attacks that are independent on the initial attack. Even if the effort can be alleviated by using virtual machines (e.g. see [47]) the honeypot is still unusable while reinstalling. Therefore, if the position of the honeypots are known, denial of service attacks against them can very easily be executed by continuous attacks. As a consequence, the position of the honeypots must be carefully hidden and known surveillance methods should be taken into account (see e.g. [29, 40, 35]).

However, only high-interaction technologies will allow to accurately follow the aims of the NoAH project presented in Chapter 1. To cope with the aforementioned problems concerning the low scalability of high-interaction honeypots, hybrid approaches have been proposed in [34], [27] and [26]. Hybrid approaches introduce a two step structure by combination of low and high-interaction components. In the first step, all traffic is processed by low-interaction components (e.g. honeyd) that allow to cover a broad range of IP addresses. In the second step, selected connections are relayed to high-interaction components where they are analysed. Thus, hybrid approaches try to combine the advantages of low as well as high-interaction components.

The authors of [27] propose to use the low-interaction components to filter out all previously unseen attacks and relay them to the network of high-interaction honeypots. This seems to be an important advantage because the aforementioned problem of the high deployment effort of high-interaction honeypots can be alleviated. Thus, all previously known attacks can be discarded. Only unknown attacks which are much more interesting to this project are analysed on the high-interaction honeypots. The detection of unknown attacks is especially critical if they are abused by a novel worm to spread (e.g. see [67]). Because the number of unknown attacks is commonly much smaller than the number of known attacks (see [54]), the effort can be significantly reduced and all effort is focused on the analysis of unknown attacks. In addition, hybrid approaches are able to alleviate the problem of denial of service attacks against high-interaction honeypots because these attacks can be blocked at the first stage. Known attacks can e.g. be identified by host based signatures as produced by Vigilante or by common IDS (e.g. *Snort*, *Bro*, or *Prelude*). In addition, as proposed in [30], the detection of known attacks can be done directly on the network card without dependency on a host or network based IDS.

It should be noted, that the acceptance of low-interaction components by third parties is much higher than the acceptance of high-interaction components, as seen from the eCSIRT and LEURRE.COM projects that both successfully deploy large-scale networks of low-interaction honeypots. Both projects have been able to

spread the network of honeypots to an increasing number of independent sites. This is in contrast to the solely deployment of high-interaction honeypots that may expose high risks to the deploying site and requires much more effort in maintenance. As shown in [67] the extend of the sensor network is critical for the early capture of worms. To establish a large sensor network following the aims of the NoAH project, it seems advantageous to distribute low-interaction components in a large scale relaying selected traffic to a smaller network of high-interaction honeypots where only unseen attacks have to be analysed.

Bibliography

[1] Bochs ia-32 emulator project. *http://bochs.sourceforge.net/*.

[2] Bro intrusion detection system. *http://bro-ids.org/*.

[3] The click modular router project. *http://pdos.csail.mit.edu/click/* .

[4] The covirt project. *http://www.eecs.umich.edu/CoVirt/* .

[5] The european csirt network. *http://www.ecsirt.net/* .

[6] Faumachine project. *http://www3.informatik.uni-erlangen.de/Research/FAUmachine/* .

[7] Honeycomb – automated ids signature creation using honeypots. *http://www.cl.cam.ac.uk/ cpk25/honeycomb/* .

[8] Honeynet project. *http://www.honeynet.org/papers/bots/* .

[9] idefense labs multipot. *http://www.idefense.com/iia/labs-software.php?show=9* .

[10] Leurre.com honeypot project. *http://www.leurrecom.org/* .

[11] The lucidic.net project. *http://www.lucidic.net*.

[12] mwcollect homepage. *http://www.mwcollect.org/* .

[13] Nfdump. *http://nfdump.sourceforge.net/*.

[14] Nfsen. *http://sourceforge.net/projects/nfsen/*.

[15] Prelude hybrid ids project. *http://www.prelude-ids.org/* .

[16] Qemu homepage. *http://fabrice.bellard.free.fr/qemu/about.html*.

[17] Sebek homepage. *http://www.honeynet.org/tools/sebek/*.

[18] Sirios - vorfallsbearbeitungssystem für computer-notfallteams. *http://www.cert-verbund.de/sirios/*.

[19] Snort ids homepage. *http://www.snort.org/*.

[20] Systrace - interactive policy generation for system calls. *http://www.citi.umich.edu/u/provos/systrace/*.

[21] Uml virtual machine. *http://dssg.cs.umb.edu/projects/umlvm/*.

[22] The user-mode linux kernel home page. *http://user-mode-linux.sourceforge.net/* .

[23] Vmware homepage. *http://www.vmware.com/* .

[24] Honeynet Project, Know Your Enemy: GenII Honeynets . *http://www.honeynet.org/papers/gen2/index.html* , May 2005.

[25] Honeynet Project, Know Your Enemy: Tracking Botnets . *http://www.honeynet.org/papers/bots/* , March 2005.

[26] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14th Usenix Security Symposium*, Baltimore, MD, USA, July 31 – August 5 2005.

[27] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos. A hybrid honeypot architecture for scalable network monitoring. Technical report cse-tr-499-04, Department of Electrical Engineering, University of Michigan, October 2004.

[28] E. Balas and C. Viecco. Towards a third generation data capture architecture for honeynets. In *Proceedings of the 6th IEEE Information Assurance Workshop*, West Point, New York, 15-17 June 2005.

[29] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In *Proceedings of the 14th Usenix Security Symposium*, Baltimore, MD, USA, July 31 – August 5 2005.

[30] H. Bos and K. Huang. Towards software-based signature detection for intrusion prevention on the network card. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, WA, September 2005.

[31] M. Costa, J. Crowcroft, M. Castro, and A. Rowstron. Can we contain internet worms? Technical report msr-tr-2004-83, Microsoft Research, Cambridge, UK, August 2004.

[32] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In *SOSP'05*, Brighton, UK, October 2005.

[33] J. R. Crandall, S. F. Wu, and F. T. Chong. Experiences using minos as a tool for capturing and analyzing novel worms for unknown vulnerabilities. In *Proceedings of the GI/IEEE SIG SIDAR Conference on Detection and Intrusions and Malware and Vulnerability Assessment (DIMVA)*, Vienna, Austria, July 2005.

[34] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen. Honeystat: Local worm detection using honeypots. In *Proceedings of the Recent Advance in Intrusion Detection (RAID) Conference 2004*, September 2004.

[35] M. Dornseif, T. Holz, and C. Klein. Nosebreak - attacking honeynets. In *Proceedings of the 5th IEEE Information Assurance Workshop*, June 2004.

[36] G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, December 2002.

[37] D. R. Ellis, J. G. Aiken, K. S. Attwood, and S. D. Tenaglia. A behavioral approach to worm detection. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, October 2004.

[38] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.

[39] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[40] T. Holz and F. Raynal. Detecting honeypots and other suspicious environments. In *Proceedings of the 6th IEEE Information Assurance Workshop*, June 2005.

[41] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of the IEEE WETICE 2003 Workshop on Enterprise Security*, Linz, Austria, June 2003.

[42] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th Usenix Security Symposium*, San Diego, CA, USA, August 2004.

[43] O. Kolesnikov, D. Dagon, and W. Lee. Advanced polymorphic worms: Evading ids by blending in with normal traffic. Technical report git-cc-04-15,, College of Computing, Georgia Tech, 2004-2005.

[44] C. Kreibich and J. Crowcroft. Honeycomb – creating intrusion detection signatures using honeypots. In *2nd Workshop on Hot Topics in Networks (HotNets-II)*, 2003.

[45] B. Krishnamurthy. Mohonk: mobile honeypots to trace unwanted traffi c early. In *NetT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting*, pages 277–282, New York, NY, USA, 2004. ACM Press.

[46] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2005.

[47] S. Nanda and T. Chiueh. A survey on virtualization technologies. Technical report tr-179, Department of Computer Science, State University of New York, February 2005.

[48] J. Newsome, B. Karp, and D. Song. Polygraph: Automatic signature generation for polymorphic worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

[49] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*, San Diego, California, USA, February 2005.

[50] A. Pasupulati, J. Coit, K. Levitt, S. Wu, S. Li, R. Kuo, and K. Fan. Buttercup: On network-based detection of polymorphic buffer overfbw vulnerabilities. In *Proceedings of the 9th IEEE/IFIP Network Operation and Management Symposium (NOMS'2004)*, Seoul, S. Korea, May 2004.

[51] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks: special issue on intrusion detection*, 31(23-24):2435–2463, December 14th 1999.

[52] J. Pincus and B. Baker. Beyond stack smashing: Recent advances in exploiting buffer overruns. *In IEEE Security & Privacy Magazine*, 2(4):20–27, 2004.

[53] F. Pouget and M. Dacier. Honeypot-based forensics. In *Proceedings of the AusCERT Asia Pacifi c Information technology Security Conference 2004*, 23rd - 27th May 2004.

[54] F. Pouget and T. Holz. A pointillist approach for comparing honeypots. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA05)*, July 2005.

[55] N. Provos. Improving host security with system call policies. In *Proceedings of the 12th USENIX Security Symposium*, Washington DC, USA, August 2003.

[56] N. Provos. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, August 2004.

[57] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fi ngerprinting. In *Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation*, San Francisco, CA, USA, December 2004.

[58] A. Smirnov and T. Chiueh. Dira: Automatic detection, identifi cation, and repair of control-hijacking attacks. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*, San Diego, California, USA, February 2005.

[59] Y. Tang and S. Chen. Defending against internet worms: A signature-based approach. In *Proceedings of the IEEE Infocom 2005*, Miami, Florida, USA, May 2005.

[60] T. Toth and C. Kruegel. Accurate buffer overfbw detection via abstract payload execution. In *Proceedings of the Recent Advance in Intrusion Detection (RAID) Conference 2002*, October 2002.

[61] N. Vanderavero, X. Brouckaert, O. Bonaventure, and B. L. Charlier. The honeytank : a scalable approach to collect malicious internet traffi c. In *Proceedings of the International Infrastructure Survivability Workshop (IISW'04)*, Lisbon, Portugal, December 2004.

[62] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the 14th Usenix Security Symposium*, Baltimore, MD, USA, July 31 – August 5 2005.

[63] K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the Recent Advance in Intrusion Detection (RAID) Conference 2004*, September 2004.

[64] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the domino overlay system. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*, San Diego, California, USA, February 2004.

[65] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In *Proceedings of the Recent Advance in Intrusion Detection (RAID) Conference 2004*, September 2004.

[66] V. Yegneswaran, J. T. Giffi n, P. Barford, and S. Jha. An architecture for generating semantics-aware signatures. In *Proceedings of the 14th Usenix Security Symposium*, Baltimore, MD, USA, July 31 – August 5 2005.

[67] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS'03)*, Washington DC, USA, October 27–31 2003.