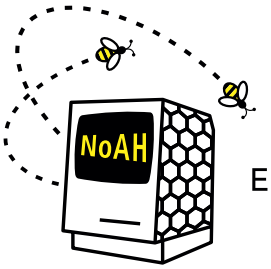
SIXTH FRAMEWORK PROGRAMME

Structuring the European Research Area Specific Programme

RESEARCH INFRASTRUCTURES ACTION



European Network of Affined Honeypots

Contract No. RIDS-011923

D3.2: Methodology and Evaluation of NoAH

Abstract: This deliverable defines a methodology for evaluating the overall advantages and possible limitations of NoAH, and it demonstrates the benefits of NoAH. The evaluation of the system focuses on multiple criteria in several dimensions including i) the detection speed of (novel) cyberattacks, ii) the accuracy of the detection methods, iii) the effectiveness of the virtual machine containment environment, and iv) the robustness of the NoAH infrastructure against malfunctions and/or malicious attacks. The obtained evaluation results proof that NoAH is a valuable tool in protecting our systems against zero-day cyberattacks. Moreover, we show that the NoAH infrastructure design is robust and scalable in nature.

Contractual Date of Delivery	31/01/2008
Actual Date of Delivery	17/03/2008
Deliverable Security Class	Public
Editor	ETHZ

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

List of Figures

List of Tables

Chapter 1

Introduction

1.1 System Overview

In Fig. 1.1 we present the whole NoAH architecture. We show the individual steps of attack processing within NoAH: The first step is the attack redirection through funnels and the anonymization framework TOR from honey@home clients or participating organizations. In the second step, known attacks are filtered on low interaction honeypots in the NoAH core, and novel attacks are forwarded to the high-interaction honeypot. The third step is about attack detection on high interaction honeypots. In the fourth step, a signature is generated automatically upon the receipt of an alert from the high-interaction honeypot. Finally, in the fifth step, the generated exploit signatures and related information are stored in the NoAH database.

1.2 Evaluation Methodology

The evaluation of the NoAH architecture is conducted in a component-wise manner by the different NoAH partners. [1] This allows us to assess the performance and robustness of each component without taking dependencies into account. The NoAH components are evaluated according to the following criteria:

- *Robustness and Stability* are evaluated for each component of the NoAH architecture.
- *Accuracy* is mainly an issue for attack detection (Argos) and signature generation (SGC) and is only evaluated for these components.
- *Throughput and Delay* are an issue, and consequently are evaluated, for the forwarding and filtering infrastructure (honey@home, honeyd, Argos), for the signature generation (SGC), as well as for the database backend.

[1] The evaluation of the router component has been shifted to deliverable D3.3.

- *CPU Overhead* is an issue, and is thus evaluated for two components, namely, honeyd and Argos.

1.3 Outline

In Chapter 2, we present a detailed evaluation of the detection accuracy of Argos, the high-interaction honeypot developed for reliable attack detection within NoAH. Argos is tested with a selection of different exploits. It is shown that Argos can reliably detect attacks that rely on memory corruption.

In Chapter 3, evaluation results for a prototype implementation of the signature generation component (SGC) are presented. We show that the SGC can automatically generate accurate signatures within an acceptable delay. Moreover, we assess the quality of the generated Snort signatures and show that the SGC is robust and scalable.

In Chapter 4, we evaluate the redirection and filtering infrastructure with respect to functionality, robustness, and achieved throughput. We show that honey@home clients are resilient to failures by conducting a 3-day stability test. We also show the trade-off between performance in terms of achieved throughput and security by introduction of anonymization with TOR. Moreover, experimental results for the CPU overhead of Argos from a test installation are presented.

Chapter 5 presents evaluation results for the database backend that is used for storing signatures and traces. This includes an evaluation of the management interface with regard to common criteria such as user friendliness, and completeness. Furthermore, the efficiency of insert statements was measured.

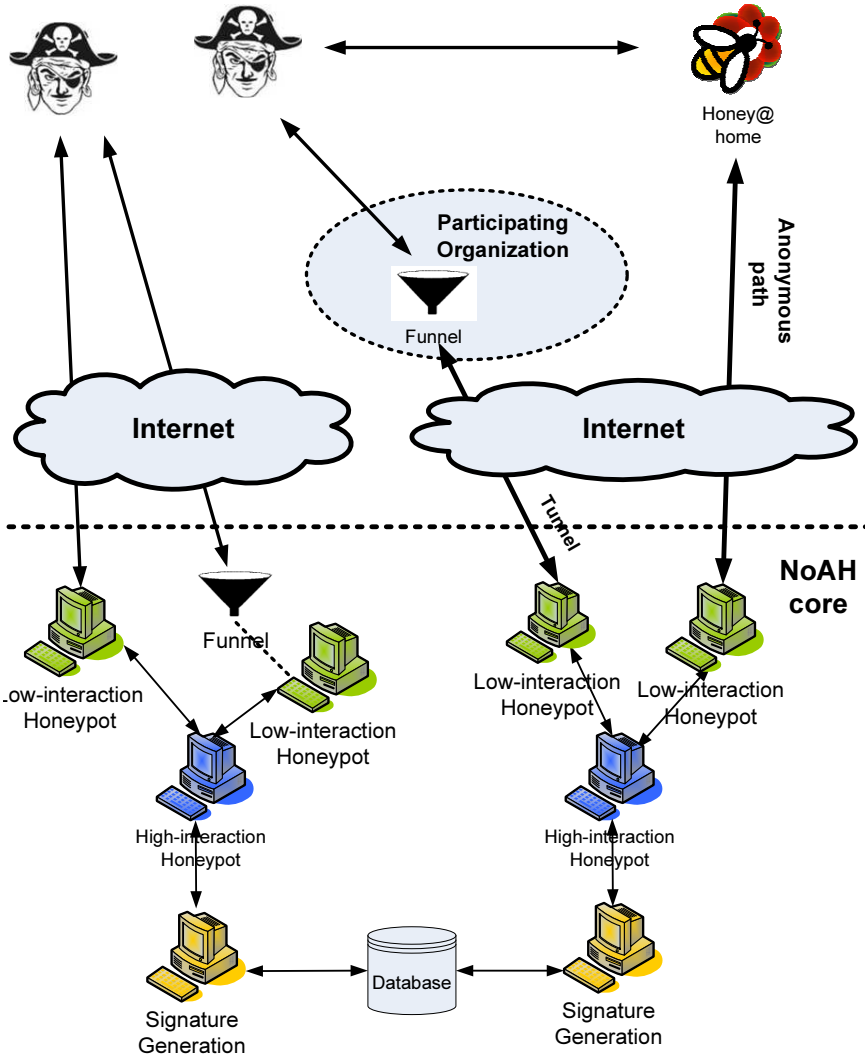


Figure 1.1: Overview of NoAH Architecture

Chapter 2

Argos Evaluation

2.1 Introduction to Vulnerabilities and Exploits

In this section, we give a brief overview of vulnerabilities and exploits which are relevant for understanding the Argos evaluation.

2.1.1 Vulnerabilities

A vulnerability is some kind of weakness which can be exploited by an attacker to get in control of a computer system. The vulnerability can affect a program, the kernel, or a component of the operating system. In general, a vulnerability in the operating system is very severe because it allows the attacker to fully control the vulnerable system. In addition, some vulnerabilities can be exploited over the network. Other require a local account on the vulnerable system. The vulnerabilities can be divided into the following classes:

Memory Corruption This class includes all programing mistakes leading to a corruption of memory structures:

- Overflow of a heap-based buffer
- Overflow of a stack-based buffer
- Integer Overflow
- Format String vulnerabilities
- Double Free Vulnerabilities
- Other: This includes e.g. the erroneous calculation of pointers, out-of-bounds access of an array variable.

Common to all vulnerabilities is that the resulting memory corruption can be abused to divert the program flow. In general, these vulnerabilities are being exploited by the injection of shell-code to which the program flow is diverted in some way. If the vulnerable buffer is on the stack segment commonly the return address is overwritten by a pointer pointing to the shell code.

Insecure Configuration An insecure configuration can also lead to the compromise of a computer system. For example, if some configuration includes standard passwords for a user account or enables some insecure options. The Cisco Application Velocity Appliances suffered from this problem (see Cisco Security Advisory cisco-sa-20080123). An additional example is the insecure standard configuration of the linux automount tool in some linux distributions which had been configured to respect the "SetUID" flag of programs (CVE-2007-5964). This could be exploited by an attacker controlling the NFS server to gain root privileges on a remote client system.

Insecure Use or Provision of Features A prominent example of this class is the vulnerability in the RDS.DataControl ActiveX Control (CVE-2006-0003) which is used by the Microsoft Internet Explorer. This ActiveX control provides the method "createObject" which can be used to invoke other ActiveX controls. A flaw in the implementation of this method invoked objects with inherited elevated privileges which allows an attacker to access the local file system.

In addition, PHP code injection and SQL injection vulnerabilities are also part of this class. PHP code injection vulnerabilities result often from an insecure use of the `include()` method. Commonly, a PHP file is included whose name is specified as a parameter in the HTTP GET request or a global variable. As a typical mistake the name of the PHP file is not sanitized or the global variable can be overwritten. SQL vulnerabilities are caused by the insecure invocation of SQL queries. This class of vulnerabilities allows an attacker to modify an SQL query to practically execute own SQL commands or queries. An attractive target is to display or modify the database table containing user credentials of the specific program, e.g. content management systems for webservers. Once the attacker knows the password of the administrator account he is e.g. able to upload phishing sites or malware on the vulnerable server.

Design Flaws Protocols and cryptographic algorithms should be designed to securely transfer data over the network. However, flaws can break the security and open holes which can be abused e.g. to get unauthorized access to systems or data. An example are flaws in the design of the SSLv2 algorithm leading to a cryptographic weakness in the algorithm.

2.1.2 Exploits

In this section an *exploit* is understood as a program or procedure to attack a vulnerability. Nearly all exploits for memory corruption vulnerabilities try to inject executable code (*shell code*) as program data and divert the control flow in such a way that the shell code gets executed. The diversion of the control flow is done by overwriting data structures of the program controlling the program flow. For example, these data structures include the return address of the current stack frame,

function pointers, and pointers of memory allocation structures (double free vulnerabilities). In general, a memory corruption vulnerability can be exploited to compromise a system if the value of the overwritten pointer can be at least partially controlled by the attacker. As a remaining problem the attacker often does not know the address at which the shell code is stored in memory. If the exploit does not lead to a crash of the program this address can be iteratively found out by brute-force guessing. A different strategy to overcome this problem can be used if either a processor register points to a location with a constant relative offset to the shell code or a function pointer exists at a constant location. In the first case a small code fragment is injected at a known memory address (often called "jump code") whose only purpose is to divert the control flow to the shell code.

Commonly, exploitation of PHP code injection and SQL injection vulnerabilities do not require any control flow diversion and are therefore more trivial than exploitation of a memory corruption vulnerability. These vulnerabilities are often exploited by the manipulation of one or more parameter in a HTTP Get or Post request. Since the attack solely proceeds on the application level (no memory corruption is involved) the risk of crashing the application is very low.

Defense strategies in modern operating systems

A buffer overflow in the `fingerd` service was abused by the famous Morris worm to break into vulnerable machines back in 1988. Buffer overflows and their consequences for computer security are well known for a quite long time. Since then, much effort has been spend into developing defense mechanisms to protect against the exploitation of memory corruption vulnerabilities. Many of these mechanisms are used in modern operating systems:

Address space randomization (ASR) Address space randomization has been introduced in the linux operating system with kernel version 2.6.12 (see [6]). The basic idea of ASR is to counter exploits for buffer overflow vulnerabilities by avoiding memory structures with constant locations in the stack segment. The addresses of the stack layout is chosen dynamically at the start of the program and prevents all exploits which rely on constant address (e.g. of the shell code) on the stack. Therefore, exploitation of buffer overflow vulnerabilities becomes more difficult with the introduction of ASR.

Canary Protection of Pointers Most often the return address of a stack frame is overwritten exploiting a buffer overflow on the stack segment. In detail, all data on the stack is consecutively overwritten until the return address is reached. A generic protection mechanism is to write a *canary word* directly in front of the return address. If a buffer is overwritten the return address cannot be overwritten without violating the canary word. Since the canary is either a random word or consists of carefully chosen characters it cannot be guessed or spoofed by the attacker. Thus, the integrity of the canary is checked each time the program leaves the corresponding subprocedure and

the program is terminated if the canary is violated. For example, this canary protection mechanism is introduced with Stackguard in the linux operating system or Service Pack 2 in Windows XP. In general, the same approach allows to assure the integrity of pointers on the heap segment.

Heap Protection The heap segment allows a program to dynamically allocate and free memory. Technically, it is implemented as a list of memory chunks that is organized as a data structure which primarily is a doubled linked list. The weakness of this structure is that the administrative data is stored in-line with the user data. As a consequence this administrative data is overwritten by user data if a buffer on the heap is overflown. An attacker can abuse this weakness by the injection of specially crafted data to overwrite any 4-byte word in the address space of the program (e.g. a function pointer). To counter this weakness, all current implementations of the heap administration (e.g. the `glibc` library) include integrity tests of the heap before doing critical operations. In Microsoft Windows heap protection has been introduced with Service Pack 2. In addition, heap integrity is enforced using a canary protection mechanism.

The introduction of these protection mechanisms raises the bar of exploiting a memory corruption vulnerability. However, none of these approaches is able to directly prevent a corruption of the memory structures of a vulnerable program and some approaches have been published to overcome the protection. In general, these approaches require a deep understanding of the memory layout and structure of the vulnerable program to find weak points which are still present. For example, Immunity has added an exploit for the heap overflow in Microsoft Windows XP 2 to the CANVAS exploit suite which is triggered by specially crafted IP packets (see [8] and [5] for further details).

2.2 Evaluation of detection accuracy

Argos runs a native operating system in a virtual machine designed to detect attacks against the guest operating system. For that purpose the monitor of the virtual machine is instrumented to detect malicious behavior with respect to the usage or execution of data originating from the network. In detail, all data which has been received by the virtual network card is marked as tainted. If tainted data is copied or modified in some way this data is also marked as tainted. If tainted data is being executed by the CPU or loaded into the instruction pointer of the CPU or used exclusively as data in system calls malicious behavior is detected and an alert is risen (see deliverable D1.4 for further details). Motivation for this procedure is on the one hand that any data originating from the network must not influence the control flow of a program. On the other hand this behavior occurs in general while exploiting a memory corruption vulnerability. Argos is based on this generic way for detecting attacks and we expect Argos to reliably detect all flavors

of memory corruption vulnerabilities. However, as pointed out at the description of vulnerabilities SQL and PHP code injection vulnerabilities are exploited on the application layer without leading to a corruption of memory structures.

The following issues are considered for the evaluation of Argos' detection accuracy:

- Are exploits for memory corruption vulnerabilities reliably detected or does this depend on the type of the memory corruption vulnerability (e.g. heap overflow vs stack based buffer overflow) or the specific exploit?
- Is Argos able to detect attacks exploiting vulnerabilities not leading to memory corruption; e.g. SQL or PHP injection vulnerabilities? If so, under which conditions?

2.2.1 Selection of Exploits

The following criteria are used for the selection of exploits:

Reliable source Unfortunately, there are only a few reliable sources for public available exploits. The primary risk is that the exploit contains a back-door which is activated at the execution of the exploit program. In addition, the information about the attacked vulnerability often tends to be wrong. For example, it is common that exploit programs are copied and slightly modified to get credits for the publication or detection of the vulnerability. For the Argos evaluation we decided to use the *Metasploit* framework ([7]), the securityfocus.com vulnerability database and the Milw0rm database of exploits ([9]).

Date of the vulnerability disclosure Exploits for vulnerabilities which have been recently disclosed are preferred.

Operation System In general, exploits are designed for specific versions of operating systems. Adaption to a specific version is commonly required because the location of memory structures often varies from one version to another. Therefore, we prefer exploits for actual operating systems which are used as honeypots in the NoAH demonstrator (Task 3.1) In addition, as pointed out in the previous section, important improvements in security protection mechanisms have been introduced in the actual versions of the linux and Windows operating system. Therefore, the exploits must change their strategy to exploit a memory corruption vulnerability.

Popularity of the Program For a realistic evaluation we focus on popular programs which are widespread. These include the Apache webserver, Microsoft Internet Explorer and basic components of the Microsoft Windows operating system (e.g. the LSASS system).

Type of vulnerability As pointed out earlier, there are different types of vulnerabilities which are exploited in different ways. This applies for the class of memory corruption vulnerabilities itself as well as other classes of vulnerabilities. It is important to note that these different types of vulnerabilities lead to a different Argos' detection patterns (e.g. results in a different Argos alert type). Therefore, consideration of different types of vulnerabilities is crucial for a comprehensive evaluation of Argos.

2.2.2 Argos alerts

If Argos detects a malicious usage of data an alert is risen. This alert consists of:

- The type of the alert. Argos distinguishes between different types of malicious behavior:

  RET The return address of a stack frame has been overwritten by tainted data. This type of alert can be expected while exploiting a stack based buffer overflow.

  JMP The address of a jmp assembly command or a related command is tainted.

  CALL The address of a call assembly command or a related command is tainted.

  CI Direct execution of tainted data.

- The state of the CPU including all values of the processor registers.
- All data blocks that contain tainted data.
- The contents of the Ethernet frame of the malicious network traffic if the Connection Tracker has been able to track down this data.

The primary result of the evaluation is if Argos correctly detects the exploitation of a vulnerability. If Argos correctly detects the attack, the type of alert and the processor state are presented for a detailed interpretation. In the other case, we argue why Argos failed.

2.3 Experimental Results

For each corresponding pair of vulnerability and exploit we first describe briefly the vulnerability itself. Details about the vulnerability are of special interest if the vulnerability falls into the class of memory corruption. For example, an exploit leveraging a stack based buffer overflow will likely overwrite the return address of the current stack frame. As a consequence, it can be expected that Argos raises an alert of type RET corresponding to this behavior. In addition, most exploits for

heap based overflows will most likely overwrite a function pointer to gain control of the program flow. If the function pointer is overwritten by tainted data an alert of type `JMP` or `CALL` can be expected.

phpBB highlight vulnerability: CVE-2004-1315

The PHP application phpBB contains a flaw in the sanitization of the `highlight` parameter. This flaw can be leveraged to inject arbitrary PHP commands which are executed in the context of phpBB.

Argos Results

This vulnerability is pure insecurity at the application layer resulting from the missing sanitization of a user supplied value in a PHP command. Since Argos is designed to detect and analyze memory corruption vulnerabilities, the exploit proceeds without detection.

Microsoft Data Access Components Vulnerability: CVE-2006-0003

The Microsoft Data Access Components contain the RDS.Dataspace ActiveX Control. This ActiveX control provides an insecure method `CreateObject` which can be used to instantiate other ActiveX controls. However, if invoked in a specific way this method does not apply any security checks to its arguments and allows to create ActiveX controls running in the local security zone of the Internet Explorer. As a consequence, these ActiveX controls have complete access to the local file system and therefore can download and execute any files on the local machine. Exploits use this vulnerable ActiveX control to download Trojan code to a vulnerable machine and to execute this code.

Argos Results

As previously pointed out, this vulnerability does not fall into the class of memory corruption vulnerabilities. Instead, it abuses missing security checks in the functionality of the ActiveX control RDS.Dataspace.

The exploit included in the *Metasploit* framework uses the vulnerability to download a Trojan binary `metasploit.exe` to the vulnerable machine. As expected, Argos did not detect this kind of attack. However, the downloaded binary `metasploit.exe` crashes with a segmentation fault. Since the exploit proceeded on an identical virtual machine running in a Qemu 0.8.2 environment, this behavior can be related to a side effect of Argos.

Windows Vector Markup Language Vulnerability: CVE-2006-4868

The Windows Vector Markup Language Library (`Vgx.dll`) was affected by a vulnerability related to the improper handling of the parameter “fill” in VML graphics.

This flaw leads to a stack based buffer overflow. It can e.g. be exploited by providing an especially crafted HTTP web page. If the user loads the web page in the Internet Explorer the vulnerability is triggered to execute arbitrary commands on the affected system. This vulnerability raised public attention because the exploit was widely abused to compromised machines before a security fix from Microsoft was available (*zero-day exploit*).

Argos Results

An exploit for this vulnerability is contained in the *Metasploit* framework. Since the exploit fills large portions of memory with shell-code, we believe that the vulnerability does not allow to gain full control over the return address of the current stack frame. Consequently Argos does not detect a manipulated return address but the execution of tainted shell-code (alert type `CI`). Note, that the address of the EIP `0x0c0c0c0c` supports this assumption.

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: NO

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            CI              1194982309

EAX             ECX             EDX             EBX
 0x00000000      0x0c0c0c0c      0x7c9137d8      0x00000000
(0x00000000)    (0x031d9ab4)    (0x00000000)    (0x00000000)


ESP             EBP             ESI             EDI
 0x0012ba7c      0x0012ba9c      0x00000000      0x00000000
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)


EIP             Faulty EIP      EFLAGS
0x0c0c0c0c      0xffffffff      0x00000202
(0x00000000)
```

OpenSSL / Apache Vulnerability: CVE-2002-0656

The vulnerability in *CVE-2002-0656* affects the data structure `ssl_session_st` in OpenSSL 0.9.6d and older versions. Since the vulnerable code is used in the Apache webserver the vulnerability also affects all Apache servers with a vulnerable OpenSSL version which accept SSL connections. Since this vulnerability has

been abused by an Internet worm, a very large number of servers were compromised during its spreading.

The vulnerability is caused by trusting the user supplied value KEY_ARG_LENGTH. By providing a spoofed value OpenSSL can be eluded to allocate a memory buffer which is too small for the SSL Client key which is provided by the attacker. As a consequence, a buffer overflow on the heap segment is triggered which can be leveraged to overwrite critical data structures of the program.

For the Argos evaluation the exploit in http://www.phreedom.org/solar/exploits/apache-openssl/ and a SuSE linux system running Apache version 1.3.24 have been selected. This exploit overwrites administrative heap data [1] and finally diverts the program control flow by the manipulation of the Global Offset Table (GOT).

Argos results

The program flow diversion of the exploit is reliably detected by the Argos honeypot. In detail, Argos detects that the address of a jmp assembly instruction is overwritten by user supplied data:

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: NO

VERSION         ARCH            TYPE            TIMESTAMP
0x01            i386            JMP             1202750084

EAX             ECX             EDX             EBX
 0x08050490      0x00000000      0x08107988      0x404d50cc
(0x00000000)    (0x00000000)    (0x02071570)    (0x00000000)


ESP             EBP             ESI             EDI
 0xbffff5a0      0xbffff69c      0x0810fab8      0x081077b0
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)


EIP             EFLAGS
0x08107990      0x00000202
(0x0211ada8)
```

[1] Note that this exploit will likely fail on modern operating systems because of the previously described heap protection mechanisms

Apache chunk-encoded HTTP Request Vulnerability: CVE-2006-3747

Apache 1.3 through 1.3.24, and Apache 2.0 through 2.0.36 include a vulnerability in the handling of malformed chunk-encoded HTTP Request (see RFC2616 for a reference of chunk-encoded HTTP Requests). As a consequence, a buffer overflow on the stack segment occurs if a malformed chunk-encoded HTTP Request is sent to a vulnerable webserver.

Argos results

For the experimental test an exploit included in the *Metasploit* framework has been used. This exploit guesses the address of the shell-code by a brute-force approach. The attack is reliably detected because of the execution of tainted shell-code (alert type `CI`).

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: NO

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            CI              1194975470

EAX             ECX             EDX             EBX
 0x00000000      0x007cdb58      0x7c9137d8      0x00000000
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)


ESP             EBP             ESI             EDI
 0x007cda7c      0x007cda90      0x00000000      0x00000000
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)


EIP             Faulty EIP      EFLAGS
0x007cffa4      0xffffffff      0x00000202
(0x00000000)
```

Apache mod_rewrite Vulnerability: CVE-2006-3747

In the mod_rewrite module of the Apache server (1.3 branch: $> 1.3.28$ and $< 1.3.37$, 2.0 branch: $> 2.0.46$ and $< 2.0.59$, 2.2 branch: $> 2.2.0$ and $< 2.2.3$) exists a vulnerability affecting the handling of URL rewrites. Due to an error in the indexing of the array `token` user supplied data can be written out of the bounds

of this array leading to a corruption of the current stack frame. However, the attacker cannot directly overwrite the return address because the boundary of the array can only be overwritten by exactly one element. The vulnerability can only be exploited if a specific rewrite rule is configured and a specific stack layout is required.

Argos results

For the test a Windows XP operating system was chosen running Apache version 2.0.59. Since the exploit triggers the vulnerability two times to execute shell code on the vulnerable machine two alerts were risen by Argos. Both alerts are caused by the execution of tainted assembly code (type `CI`):

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: YES

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            CI              1198310757

EAX             ECX             EDX             EBX
 0x005d29c8      0x00000000      0x005d29ea      0x00000001
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[     98255]    [     98251]    [     98235]    [     98215]

ESP             EBP             ESI             EDI
 0x0452fac4      0x77c438c0      0x005cff90      0x00000007
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[     98215]    [     98215]    [     98215]    [     98215]

EIP             Faulty EIP      EFLAGS
0x005d26e2      0xffffffff      0x00000202
(0x00000000)

carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: YES

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            CI              1198310760

EAX             ECX             EDX             EBX
 0x00000158      0x0008a350      0x00000001      0x0452e818
```

```
(0x006a8808)    (0x00000000)    (0x00000000)    (0x00000000)
[     99979]    [     99979]    [     99979]    [     99979]

ESP             EBP             ESI             EDI
 0x0452e414      0x0452f9b4      0x0452f9c8      0x0000035c
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[     99979]    [     99979]    [     99979]    [     99979]

EIP             Faulty EIP      EFLAGS
0x0452e818      0xffffffff      0x00000202
(0x00000000)
```

Windows LSASS Vulnerability: CVE-2004-0533

The Active Directory service functions in LSASRV.DLL of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows NT 4.0 SP6a, 2000 SP2 through SP4, XP SP1, Server 2003 are vulnerable to a buffer overflow. This vulnerability is well known because of the W32.Sasser.Worm which leveraged it in 2004 for the spreading of the worm. The buffer overflow is caused by the unsafe use of the `vsprintf()` function to copy user supplied data into a buffer without appropriate bounds checking. This results in a buffer overflow in the stack segment.

Argos results

An exploit in the *Metasploit* framework has been used. Although this exploit fails for the target "Windows XP", it leads to a stack corruption after choosing the target "Windows 2000". Consequently, Argos raised an alert of type `RET` pointing to an return address being overwritten by tainted data.

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: YES

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            RET             1194958990

EAX             ECX             EDX             EBX
 0x00000000      0x77e79e1b      0x745b30e0      0x008afd70
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[      2595]    [      2595]    [      2595]    [      2595]
```

```
ESP             EBP             ESI             EDI
 0x008af89c      0x00790059      0x00000023      0x002652f0
(0x00000000)    (0x064d0888)    (0x00000000)    (0x00000000)
[      2595]    [      5285]    [      2595]    [      2595]

EIP             Faulty EIP      EFLAGS
0x002f0052      0x745937d8      0x00000202
(0x064d088c)
```

Windows DCOM Vulnerability: CVE-2003-0352

The RPC interface implementing the Distributed Component Object Model services (DCOM) in Windows 2000 and XP before SP 2 suffered from a buffer overflow vulnerability. In detail, an overlong user supplied file name (`szName`) leads to an overwrite of a buffer on the stack segment. Since this vulnerability was abused by the wide-spread Blaster and Welchia Internet worms it had a critical impact for the security of the Internet.

Argos results

An exploit in the *Metasploit* framework has been used. As being expected, Argos detected that the return address of a stack frame was overwritten by tainted data (alert type `RET`).

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: YES

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            RET             1194963783

EAX             ECX             EDX             EBX
 0x00000000      0x000acdd0      0x00085aea      0x005bfd18
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[       637]    [       637]    [       637]    [       637]

ESP             EBP             ESI             EDI
 0x005bf7bc      0x19eb10eb      0x000bb310      0x000bc7fc
(0x00000000)    (0x067957ac)    (0x00000000)    (0x00000000)
[       637]    [       669]    [       637]    [       637]

EIP             Faulty EIP      EFLAGS
```

```
0x0018759f      0x75879a8d      0x00000202
(0x067957b0)
```

Windows ASN.1 Library vulnerability: MS04-007

Two vulnerabilities (CAN-2003-0818 and CVE-2005-1935) related to the handling of BER encoded data were found in the Windows ASN.1 Library both leading to a corrupted heap layout. The ASN.1 library provides basic services concerning a transparent encoding of data and is used by a large number of Windows services. Therefore, this vulnerability is exposed by multiple vectors including services using the SMB, HTTP, and SMTP protocols.

Argos results

An exploit in the *Metasploit* framework has been used. This exploit is rather complex and relies on the construction of a specific heap layout leading to an overwrite of a function pointer. Argos detected the attack because of the execution of tainted shell-code (alert type `CI`).

```
carlog v0.1.2 Copyright(c) G Portokalidis

Net tracker data: YES

VERSION         ARCH            TYPE            TIMESTAMP
0x02            i386            CI              1194970995

EAX             ECX             EDX             EBX
 0x7ffdf000      0x000cfe58      0x0a28002e      0x7ffdf000
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[     14818]    [     14818]    [     14818]    [     14818]

ESP             EBP             ESI             EDI
 0x00c2f0d0      0x00c2f10c      0x00000000      0x00000000
(0x00000000)    (0x00000000)    (0x00000000)    (0x00000000)
[     14818]    [     14818]    [     14818]    [     14818]

EIP             Faulty EIP      EFLAGS
0x000cfe58      0xffffffff      0x00000202
(0x00000000)
```

2.4 Summary

The accuracy of the attack detection is crucial for the aims of the NoAH project and is consequently analyzed in this chapter. Our proceeding was to chose different public available exploits for a selected list of vulnerabilities and to analyze the corresponding results of the application of Argos.

Basically, the vulnerabilities fall into a limited list of categories consisting of memory corruption, php code inclusion and other vulnerabilities like unsafe features of ActiveX controls. Since Argos is designed to detect the control flow diversion caused by the corruption of memory structures we did not expect any positive results in respect to the non-memory corruption classes of vulnerabilities. Because all corresponding exploits proceeded without being noticed we can confirm this assumption. However, it pointed out that although the exploit was not detected by Argos, one exploit failed because of a side effect of Argos. Overall, we propose to limit the exposure of these classes of vulnerabilities by carefully choosing the application running on the honeypots. In addition, additional protection mechanism like inverse firewalls seem to be necessary to avoid an unintended compromise of the honeypot.

All attacks aiming at the exploitation of memory corruption attacks were reliably detected. Most of these exploits have been detected as soon as they manipulated a critical data structure (e.g. the return address of the current stack frame). All other attacks failed because of the execution of injected shell-code which was prevented by Argos. Although an exhaustive list of exploits was out of the scope of this evaluation, we believe that Argos will nearly detect all attacks leveraging memory corruption. However, a theoretical chance exists to write an exploit in such a way that the honeypot can be compromised only by the manipulation of program data (e.g. by a well directed manipulation of the password file). To counter this thread we propose to protect the honeypots by additional mechanisms.

Chapter 3

Signature Generation Component

We evaluate the Signature Generation Component (SGC) with respect to qualitative (i.e., functionality of the SGC and applicability of generated signatures to detect cyber attacks) and quantitative (i.e., the signature generation delay and the robustness and scalability of the SGC) criteria.

To refresh the reader's knowledge about SGC we very briefly summarize its architecture. SGC consists of two components: the protocol tracker that records the protocol connection states, and the interface that processes alerts obtained from Argos to generate meaningful signatures. For details please refer to the deliverable D1.4 on Architecture Integration [2].

3.1 Methodology

We base our evaluation on a prototype implementation of the SGC (see [3]). This SGC prototype, together with Argos/Qemu[1] and a mysql database for storing generated signatures are installed on a computer that runs Ubuntu 6.10 OS with Kernel version 2.6.17-11. This victim host has an AMD Athlon 2GHz processor with 1GB RAM and 512 KB Cache and is connected to the attacking host via an Ethernet crossover cable. Metasploit v3 and wireshark are installed on the attacking host that runs Ubuntu 7.10 OS (Kernel 2.6.22-14) and has an AMD Athlon 2GHz processor with 1GB RAM and 512 KB Cache.

We use the Metasploit framework [4] on the attacking host to generate traffic with specific exploits and run these against vulnerable applications running on the victim host (with Microsoft Windows 2000 SP4 as operating system). If the attack is successful and Argos detects it, the SGC generates a Snort signature (in one of three available formats) and stores the generated signature in the database. This setup is used to evaluate the functionality of the SGC, to measure the signature

[1] Argos 0.2.3 and Qemu 0.8.2

generation delay, and to stress test the SGC. Furthermore, we quantitatively assess the performance of generated signatures in each of the three formats.

3.2 Functionality of the SGC

The SGC collects all available information from Argos, the state tracker, and the NetBee library [10], and generates signatures in one of three available formats depending on the availability of protocol information for a particular exploit.

Signatures with protocol context and state knowledge are generated if the state tracker and the NetBee library both support the exploited protocol. These signatures specify the protocol state in which the vulnerability can be exploited, as well as the protocol context that contains the vulnerability (e.g., a field length). The required protocol state machines are implemented in the state tracker part of SGC[2] Knowledge of protocol context is implemented by the NetBee[3] library.

Signatures with protocol context knowledge are generated if the NetBee library supports the exploited protocol, but the state tracker does not. These signatures specify only the protocol context that contains the vulnerability (e.g., a field length). Knowledge of protocol context is implemented by the NetBee library.

Longest common substring signatures are generated as a backup solution if the exploited protocol is neither supported by the protocol tracker, nor by the NetBee library.

In the following, we present the Snort signatures that have been generated with our SGC prototype. For each exploit we give the generated Snort signature, the Metasploit exploit that was used, the version of the vulnerable software that was used on the victim host, and the type of signature that was generated (see Table 3.1).

The evaluation results show that the SGC works very reliably, as long as Argos provides it with the correct information about the malicious network packets that triggered the exploit. Signature generation was successful for all tested exploits.

3.3 Signature Generation Delay

We measure the signature generation delay as the time between the receipt of an alert from Argos, and the time when the generation of a Snort signature is finished and ready to be stored in the database. To get more detail, we further divide the signature generation delay in the time needed by the NetBee library for initialization and extraction of relevant protocol context information, and the time required for extracting the relevant information from the state tracker and for generating the Snort signature.

[2]The prototype currently supports one application protocol (FTP) for illustration purposes.

[3]NetBee currently supports 64 widely-used protocols.

Metasploit exploit	Vulnerable software	Generated Snort signature	Signature type
warftpd_165_user	WarFTP v1.65	alert tcp any any $->$ any 21 (flow: from_client; content: USER ; offset: 0; depth: 5; content: !|0D 0A|; offset: 5; depth: 485;)	protocol state and context aware
warftpd_165_pass	WarFTP v1.65	alert tcp any any $->$ any 21 (flow: from_client; content: "PASS "; offset: 0; depth: 5; content: !"|0D 0A|"; offset: 5; depth: 474;)	protocol state and context aware
cesarftp_mkd	Cesar FTP v0.99g	alert tcp any any $->$ any 21 (flow: from_client; content: "MKD"; offset: 0; depth: 4; content: !"|0D|"; offset: 8; depth: 670;)	protocol state and context aware
ypops_smtp	YPOPs POP3 v0.6	alert tcp any any $->$ any 25 (flow: from_client; byte_test: 2, =, 0, 4294967276; content: !"|0D 0A|"; offset: 0; depth: 978;)	protocol context aware
minishare_get_overflow	Minishare v1.4.1	alert tcp any any $->$ any 80 (flow: from_client; byte_test: 2, =, 0, 4294967276; content: "|5A 6A 71 49 31 41 72 49 65 46 49 67 6B 76 56 4F AB A3 54 77 93 22 F9 75 74 35 4B 3C 2A D4 B8 B0|";	longest common substring

Table 3.1: Exploits, vulnerable software, and generated signatures

For each type of signature, we select one representative exploit from the table presented above: warftpd_165_user for protocol state and context aware signatures; ypops_smtp for protocol context aware signatures; minishare_get_overflow for longest common substring signatures. The measurement results are presented in Table 3.2.

The average signature generation delay for the warftpd_165_user exploit is 3.22 seconds with a standard deviation of 0.48 seconds. The generation of a signature for the ypops_smtp exploit was considerably faster. It took 1.64 seconds on average with a standard deviation of 0.77 seconds. This is due to the fact that the NetBee library was considerably faster for this protocol. A signature for the minishare_get_overflow was generated on average within 3.49 seconds with a standard deviation of 0.77 seconds. This relatively large delay is due to the fact that even though the tracker and NetBee do not support the minishare protocol, they have to be queried to obtain a negative result.

As you can see in Table 3.2, the NetBee library takes a considerable part of the signature generation delay (approximately 98%) in all three cases[4]. The time for extracting information from the tracker and generating the signature is in all three examples around 0.04 seconds. Hence, the NetBee library clearly is the performance bottleneck of this system.

3.4 Robustness and Scalability

We consider two stress scenarios to test robustness and scalability of the SGC: (a) many open connections, and (b) one connection with a very high packet load. To simulate scenario (a), we generate a SYN Storm with nmap and embed the attack traffic into it. Nmap simply sends a TCP SYN packet to every port on the victim machine. This causes the state tracker to create a state machine for every initiated connection. Concurrently, we send the `warftpd_165_user` exploit to the victim. To simulate scenario (b) we embed the attack traffic in a FTP transfer of a Linux DVD image file.

In both scenarios, the correct signature could be generated - although with a larger delay due to the increased load on the state tracker which runs on the same machine as the NetBee library and the interface. In an application scenario, however, we do not expect to see such high traffic loads on the SGC since all benign traffic is filtered by the low-interaction honeypots (see NoAH deliverable D1.1 [1] for details).

3.5 Signature Quality Assessment

In this Section we assess the quality of the three types of signatures that can be generated.

[4]Most of this time is actually used for initialization of the NetBee library which takes place every time

Measurement	Average	Standard Deviation	Percentage
Protocol state and context aware signatures			
Total Time	3.22 sec	0.48 sec	100%
NetBee	3.19 sec	0.47 sec	98.92%
Signature generation	0.04 sec	0.02 sec	1.08%
Protocol context aware signatures			
Total Time	1.64 sec	0.77 sec	100%
NetBee	1.6 sec	0.77 sec	97.11%
Signature generation	0.04 sec	0.05 sec	2.89%
Longest common substring signatures			
Total Time	3.49 sec	0.77 sec	100%
NetBee	3.46 sec	0.78 sec	99.02%
Signature generation	0.03 sec	0.02 sec	0.98%

Table 3.2: Signature generation delay measurements

Signatures with protocol context and state knowledge. We expect this type of signature to produce none or very few false positives due to the fact that the signature is really trimmed to the vulnerability that is exploited, and additionally it is bound to a specific protocol state. We also expect that this type of signature does not produce any false negatives related to polymorphic payloads since it does not consider byte patterns for generating an alert.

Signatures with protocol context knowledge. We expect this type of signature to generate few false positives since the signature triggers only if the protocol specification is violated in some way (e.g., through an exceeded field length). In some cases the signature might trigger an alert even if the exploit is not successful. False negatives related to polymorphic payloads are also very unlikely since this type of signature does not rely on byte patterns.

Longest common substring signatures This fall-back signature type provides no protection against false negatives related to polymorphic payloads. It is also quite prone to false positives depending on the frequency of the substring that is used in benign traffic.

3.6 Summary

Automated signature generation is only useful if the generated signatures are accurate, and if they can be generated within a reasonable time. Regarding the accuracy the signatures we have argumentatively shown that the signatures we generate are very good, at least as long as the underlying protocol is supported. For protocol support we depend on the open-source NetBee library which is continuously improved.

Our experiments show that the signature generation delay is acceptable. We have shown that more than 95% of the delay are due to the NetBee library. Hence, an improvement of the overall delay would have to consider the NetBee library first. Also for performance reasons we suggest to run Argos and SGC on two different computers.

Chapter 4

Honey@home and honeyd Evaluation

They Honey@home architecture is comprised of four main components: the Honey@home clients, the SSL_server that handles connections from clients, honeyd that performs the network emulation and finally the high-interaction honeypots which run instances of Argos and emulate various services. For all four components we have conducted a number of experiments to evaluate their performance and stability. In this Section we will present our methodology for evaluating the Honey@home components along with some preliminary results.

4.1 Honey@home and SSL_server evaluation

The evaluation of Honey@home was conducted together with that of the SSL_server as they are tightly coupled. Honey@home cannot function as a standalone tool, without any connection endpoint. All traffic received from a Honey@home client is forwarded to the SSL_server and the latter sends the responses back to the clients. The evaluation procedure of both components was designed with 3 evaluation metrics in mind: correctness, stability and throughput. All of our experiments were done for both Windows and Linux versions of Honey@home.

Checking the correctness of the whole architecture is fairly simple. By correctness we mean that transactions performed through Honey@home clients do not yield different results than the ones performed directly from the corresponding servers. We conducted a number of transactions for three different protocols: HTTP, SSH and FTP. For all three cases, we transferred files and then compared them to the original files for possible differences. All transfers were conducted successfully, thus verifying our architecture's correctness.

In order to measure the stability of Honey@home, we ran tens of test clients and recorded their behavior. We focused on how clients behave in terms of failure. Failure can be caused by network absence, clients being killed abnormally (e.g. by sending them the KILL signal) or the anonymization network's failure to deliver

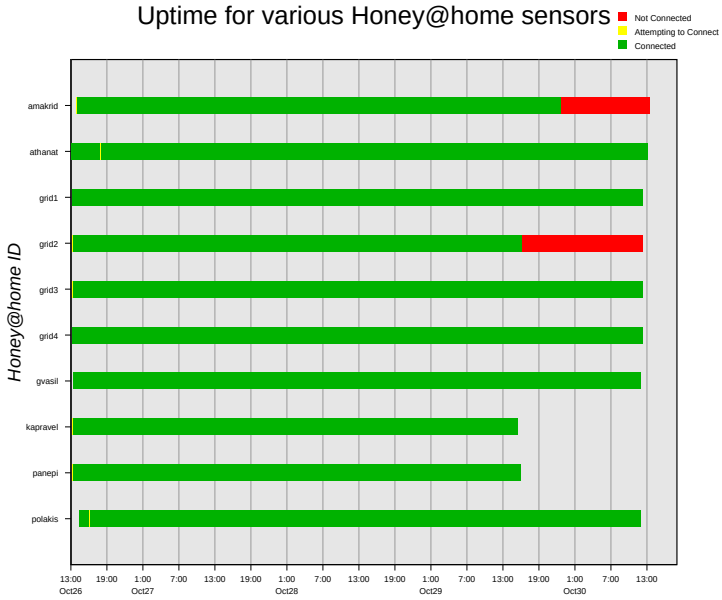


Figure 4.1: Uptime of ten Honey@home clients for a period of 3 days

our packets. Clients are designed to automatically reconnect to the SSL_server after a failure has been detected and thus their downtime is ideally small compared to their uptime. Furthermore, a heartbeat mechanism has been implemented to help clients and SSL_server identify their connection status. Our evaluation metric for stability is the uptime of clients. The SSL_server is a component with long uptime periods and closely monitored, so we focused more on the clients. We parsed the logs of our deployed clients and plotted their uptime. The client logs keeps track of all events that may occur: connections, disconnects and software failures. An example of a 3-day stability test for 12 test clients can be seen at Figure 4.1. The green color means the client is connected, yellow means it tries to reconnect and red means the client is disconnected. Ideally, the red color should not exist but clients need some time until they realize their connection with the SSL_server is lost. This is due to the fact that anonymization network does not provide us any immediate feedback in the case of a connection loss. The results displayed in Figure 4.1 are for clients with heartbeating mechanism disabled, thus some red areas exist. Our latest results for clients with heartbeat mechanism enabled show few or even none of these red areas.

Our final metric of interest is throughput. We measured the throughput of a HTTP transfer through Honey@home clients. The web server we used was Apache version 2.0. We calibrated our results by measuring the throughput of a transfer with and without Honey@home and with vanilla and Argos instances of Apache server. We also tested Honey@home clients with the feature of the anonymization

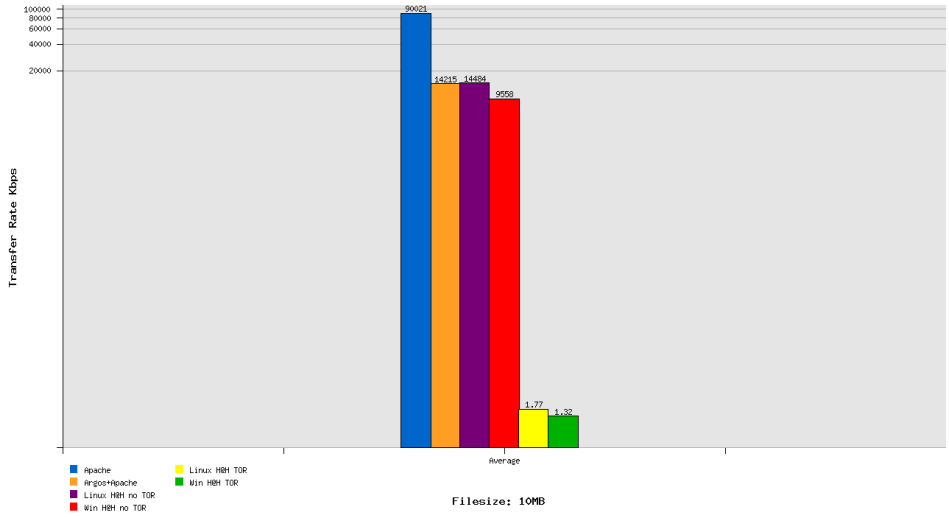


Figure 4.2: Throughput of a HTTP transfer by contacting a vanilla Apache, an Apache inside Argos and through Windows/Linux Honey@home clients with and without TOR

network disabled, so as to measure the overhead imposed by anonymous routing. All runs included a transfer of a 10Mbyte file and each number is the average of ten runs. The results are summarized in Figure 4.2. Running Apache outside a virtual machine yields a throughput of around 87 Mbps. When Apache runs inside Argos, we can achieve a throughput of around 15Mbps. If we use Honey@home clients without Tor, we achieve a throughput close to the maximum we can get, that is the one of Apache running inside Argos. We should note that our test clients reside in the same network as the SSL_server and Argos (in fact, 1 hop away) and thus the forwarding delay is minimal. We are in the process of measuring throughput from clients that are geographically dispersed. In the presence of Tor, we obtain a much lower throughput, in the order of few Kbps. This is mainly due to the fact that server anonymity requires our packets to travel through two anonymization paths (from client to a rendez-vous point and from the rendez-vous point to the SSL_server).

4.2 Honeyd

Honeyd is a well known tool used for emulating a single (or network of) “virtual” hosts running a number of different services. We have modified Honeyd (mhoneyd) by adding handoff functionality. The handoff mechanism redirects established connections to the high-interaction honeypots.

Our metrics of interest for honeyd are stability and throughput. For stability, we ran a honeyd that emulates a /8 network and flooded it with connections destined

for random hosts in the emulated network. Honeyd did not present any malfunctions and established all connections. Furthermore, its CPU and memory overhead was quite low; around 5-10% CPU overhead and 50 Mbytes of memory footprint. In the case of throughput, we were interested in measuring whether the handoff mechanism affects the throughput of Honeyd and in which cases. We tested honeyd separately from Honey@home so as to isolate any performance problems. We configured Honeyd to handoff all incoming connections to an Apache server running inside Argos. Our results showed that handoff mechanism does not affect the performance of Honeyd as we achieved a throughput of 15Mbps, the maximum an Apache inside Argos can deliver. As a side note, during our performance tests we discovered that Honeyd treats delayed acknowledgments incorrectly. That issue is now corrected and the honeyd community has been notified.

4.3 Argos

The high-interaction honeypots that comprise the NoAH core run the Argos containment environment. Argos is based on the Qemu virtual machine with some additional functionality that tracks tainted data and prohibit it from getting executed once an exploit has succeeded.

We evaluated the latest version of Argos in terms of performance. The detection accuracy of Argos is studied in 3. In Section 4.1, we highlighted its network performance and showed that a web server running inside Argos can achieve up to 15Mbps throughput. We are also interested in the CPU overhead imposed by Argos under conditions of heavy load. We monitored the CPU usage of the machine hosting Argos for a period of one hour. The results are summarized in Figure 4.3. We observe that when Argos emulates the Windows XP operating system, there is a sustained 100% CPU overhead, a fact that does not apply in the case of Linux. Nevertheless, Argos performance is not so critical since we do not expect very high loads on the high-interaction honeypot. Moreover, we are still working on the Argos performance. Lately, we managed to speed up Argos by making it switch between Xen and Argos when needed.

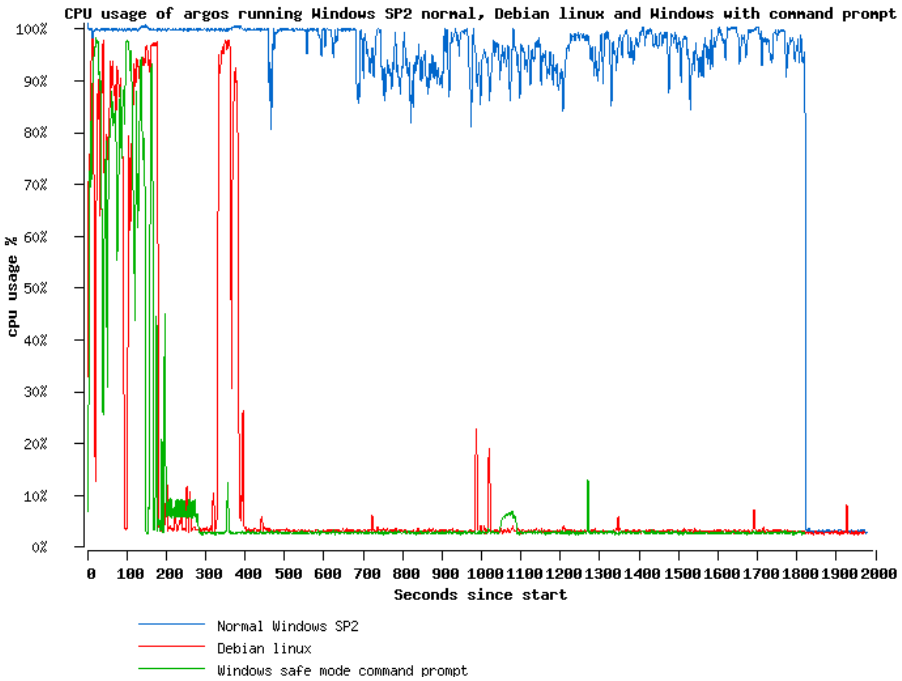


Figure 4.3: CPU utilization of Argos when it emulated Windows XP SP2 and Debian Linux operating systems

Chapter 5

Database Evaluation

While designing a database is not a difficult task for small and medium sized databases, it is an important one. Bad database design can lead to an inefficient and possibly unreliable database system. Because client applications are built to work on specific parts of a database, and rely on the database design, a bad design can be difficult to revise at a later date.

Strict database structure can be beneficial, because it can eliminate inconsistencies, such as a department with two managers. On the other hand, the designer should make designs flexible enough to allow some expansion for unforeseen uses. Extending a well-designed database is usually not too difficult, but modifying the existing table structure can render an entire database and its client applications obsolete.

The system that built is based on a mysql database with a total of 21 tables of which 5 are for the user management, 5 are for the honeypot management and the remaining tables are for Argos log storage. The front end was built on Ruby on Rails and its purpose is to configure the honeypots of the NoAH project.

5.1 Management Evaluation

To make a user interface there are several rules and aspects that the tool and the system in general, should follow:

- Speed
- Reliability
- Designed for everyone
- Help messages
- Error messages
- Compulsory fields should be pointed clearly

- Nice look and feel
- Sessions

To ensure the above aspects, exhausted tests were done on the user interface platform. We tested all the different situations that a user can come up with and below are some comments.

- When in insert or update situation compulsory fields should be pointed out.
- When in insert sensor page there should be a link point to insert new location page.
- In home page of the tool information about the system requirement are not needed.
- There is no clear relation of Noahif and NoAH project. The general template should host the NoAH logo or at list on the home page, more information about the NoAH project should exist
- In the login page if the user is not logged in the links legal and contact are not working.
- When opening a link location it should be done on a new tabbed panel and not on the same (especially, when someone is logged in).
- The honeypots will be distributed around the world and there is way to distinguish which company or organization owns what. It is recommended to add a new field to the location area describing the company or the organization.
- We don't have anywhere information of what software runs on its honeypot (nepenthes, honeyd, argos).

5.2 Database Evaluation

The relationships in the database are an encoding of rules or practices that govern the data in the entities. Once a relationship is built into the structure of the database, there is no provision for exceptions. Duplicate identifiers are not allowed.

5.2.1 Data integrity tools

To ensure the validity of data in a database, checks needs to be formulated to define valid and invalid data, and design rules to which data must adhere (also known as business rules). Together, checks and rules become constraints.

Constraints that are built into the database itself are more reliable than constraints that are built into client applications or that are spelled out as instructions to database users. Constraints built into the database become part of the definition

of the database itself, and the database enforces them consistently across all applications. Setting a constraint once in the database imposes it for all subsequent interactions with the database.

In contrast, constraints built into client applications are vulnerable every time the software changes, and may need to be imposed in several applications, or in several places in a single client application.

5.2.1.1 Defaults

Can be assigned default values to columns to make certain kinds of data entry more reliable. For example:

A column may have a current date default value for recording the date of transactions with any user or client application action. Other types of default values allow column values to increment automatically without any specific user action other than entering a new row. With this feature, you can guarantee that items (such as purchase orders for example) are unique, sequential numbers.

The following example, from an existing noahdb table, shows how the default parameter is applied on the rows. It can be null, a standard text, empty string and so on.

```
CREATE TABLE  ‘honeypot_services‘ (
‘id‘ int(11) NOT NULL auto_increment,
‘honeypot_id‘ int(11) default NULL,
‘name‘ varchar(255) default NULL,
‘transport_protocol‘ varchar(255) default ’TCP’,
‘transport_port‘ int(11) default NULL,
‘application_name‘ varchar(255) default NULL,
‘application_version‘ varchar(255) default NULL,
‘created_at‘ datetime default NULL,
‘updated_at‘ datetime default NULL,
PRIMARY KEY  (‘id‘)
)
```

5.2.1.2 Entity integrity

It keeps track of the primary keys. Primary key values are unique and cannot be null or undefined. They identify each row in a table uniquely, it guarantees that it is not null and improve the performance of the database server.

5.2.1.3 Referential integrity

It keeps track of the foreign keys that define the relationships between tables. It guarantees that all foreign key values either match a value in the corresponding primary key or contain the NULL value if they are defined to allow NULL.

5.2.2 Indexes

Performance is an important consideration when designing and creating databases. Indexes can dramatically improve the performance of statements that search for a specific row or a specific subset of the rows. On the other hand, indexes take up additional disk space and may slow inserts, updates, and deletes.

An index provides an ordering on the rows in a column or columns of a table. An index is like a telephone book that initially sorts people by surname, and then sorts identical surnames by first names. This ordering speed up searches for phone numbers for a particular surname, but it does not provide help in finding the phone number at a particular address. In the same way, a database index is useful only for searches on a specific column or columns.

Indexes get more useful as the size of the table increases. Therefore it is not necessary to be applied for our purposes but if the application starts to increase in size then the usage of them will be inevitable.

5.2.3 Procedures and Triggers

Procedures and triggers standardize actions performed by more than one application program. By coding the action once and storing it in the database for future use, applications need only call the procedure or fire the trigger to achieve the desired result repeatedly. And since changes occur in only one place, all applications using the action automatically acquire the new functionality if the implementation of the action changes.

Procedures and triggers used in a network database server environment can access data in the database without requiring network communication. This means they execute faster and with less impact on network performance than if they had been implemented in an application on one of the client machines.

5.2.4 Transactions

A transaction is a logical unit of work. Each transaction is a sequence of logically related commands that accomplish one task and transform the database from one consistent state into another. The nature of a consistent state depends on your database.

The statements within a transaction are treated as an indivisible unit: either all are executed or none is executed. At the end of each transaction, you commit your changes to make them permanent. If for any reason some of the commands in the transaction do not process properly, then any intermediate changes are undone, or rolled back. Another way of saying this is that transactions are atomic.

Grouping statements into transactions is key both to protecting the consistency of your data (even in the event of media or system failure), and to managing concurrent database operations. Transactions may be safely interleaved and the completion of each transaction marks a point at which the information in the database is consistent.

5.2.5 Database Efficiency Measurements

We made the following measurements tests with the same Argos log file (lsass_ms04_011) of size 268.5 Kbytes. There were used a client and a server both on the same LAN with the following characteristics:

Client: IBM notebook with 512M Ram, Intel Pentium 1500MHz M processor running Gentoo Linux Server: Dual Opteron 250 32bit, 4G RAM running Linux Gentoo 2.6.17

When the noahdb first installed on the client with the default options the time taken to parse the file and insert the data in the mysql database (hosted on server) was 11.45 min. This time was unacceptable and looking into it we found that the problem was due to the engine (INNODB). By removing that specific entry from the create_db.mysql file so that mysql forced to use the default engine the new improved time was 31.806 sec.

Extracting from the noahdb tool the queries it took 28.739 sec to insert the data in the database (approximately 60,000 rows). Further investigating the efficiency we extracted from the create statements of the table all the foreign keys and we had further reduced the time taken to 25.249 sec. By removing the constrains means that further checks should be made through the parsing stage of the execution. For this type of application is not recommended since it increases the programming complexity and may limit us for further expansions.

We then tested the efficiency introducing another database from another vendor (Sybase 12.5) keeping the same database schema and on the same server. The improved time was 6.3% but keeping in mind that Sybase database is not free it is not recommended for this stage.

5.2.6 Remarks

On the following table there is a unique key applied on username means that no other user can share the same username with an existing one even though the existing user may be inactive. Therefore a status_id field is recommended in order to distinguish the status of the user (active, disabled, inactive).

```
CREATE TABLE  ‘users‘ (
‘id‘ int(11) NOT NULL auto_increment,
‘username‘ varchar(255) NOT NULL,
‘password_salt‘ varchar(255) NOT NULL,
‘password_hash‘ varchar(255) NOT NULL,
PRIMARY KEY  (‘id‘),
KEY ‘index_users_on_username‘ (‘username‘)
)
```

While installing the mysql database it should be done without ssl support otherwise errors will come up on the noahdb tool compilation.

Chapter 6

Conclusion

This deliverable evaluates all components of the NoAH architecture. We have run intensive tests on prototype installations of all components. We have shown that all components basically fulfill the requirements with respect to functionality, robustness and scalability. Nevertheless, we have identified several issues that can be improved to make the system even faster, easier to manage, and more reliable.

In particular, we have shown that Argos can reliably detect a broad variety of memory corruption attacks, which it was designed for. However, the Argos approach fails on other types of attack such as PHP injection (see Chapter 2 for details). Regarding performance, Argos can achieve an acceptable maximum throughput of 15Mbps for a HTTP transfer. The CPU overhead of Argos highly depends on the emulated operating system, and is nearly 100% for Windows XP SP2.

Evaluation of the signature generation component has shown that accurate signatures can be generated within an acceptable delay if Argos has identified the network packet containing the attack correctly. The quality of the generated signatures is reasonable if the attacked protocol is supported either by NetBee or by the state tracker. This is currently the case for 64 frequently-used protocols.

The honey@home and SSL_server evaluation showed that both components work reliably over longer periods and can recover autonomously from failures. The evaluation also shows that the security improvement gained by utilizing TOR for anonymization has an impact on the throughput. The throughput with anonymization is still at a reasonable level of around 15 Mbps.

Evaluation of Honeyd has shown that this component works very reliably even when flooded with connection attempts. Moreover, the CPU and memory overhead of honeyd is small: Around 5-10% for CPU and 50 Mbytes for memory. The incorrect treatment of delayed acknowledgements that has been discovered during the evalaution has been resolved immediately. Also it has been shown that the implemented extension to honeyd, the connection handoff to Argos, does not affect the performance of honeyd.

The database evaluation has identified several issues regarding the user friendliness of the database management interface. For example, compulsory fields should

be marked as those, and some links are not working properly. Efficiency measurements of the database have identified an issue with the insertion of Argos log files. This issue was resolved by switching to a different database engine. The insertion of approximately 60.000 entries including the generation of primary keys and indexes takes about 28.739 seconds.

All in all, this evaluation has shown that the NoAH architecture can achieve it's goals: Generation of early warnings for ongoing Internet attacks that are based on memory corruption, provision of automatically generated Snort signatures for ongoing attacks with small delay, and being robust and scalable even in scenarios with high load.

Bibliography

[1] D1.1: Honeypot node architecture. *http://www.fp6-noah.org/publications/deliverables/D1.1.pdf*.

[2] D1.4: Architecture integration. *http://www.fp6-noah.org/publications/deliverables/D1.4.pdf*.

[3] D2.2: Prototype implementation. *http://www.fp6-noah.org/publications/deliverables/D2.2.pdf*.

[4] J. C. Foster. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Syngress Publishing, 2007.

[5] Immunity.com. Exploit for ms08-001. `http://www.immunityinc.com/documentation/ms08_001.html`.

[6] LWN.net. Address space randomization in 2.6. `http://lwn.net/Articles/121845/`.

[7] Metasploit. `http://www.metasploit.com/`.

[8] Microsoft. Vulnerabilities in windows tcp/ip could allow remote code execution (941644). `http://www.microsoft.com/technet/security/bulletin/MS08-001.mspx`.

[9] Milw0rm. `http://www.milw0rm.com/`.

[10] F. Risso and M. Baldi. Netpdl: an extensible xml-based language for packet header description. *Comput. Networks*, 50(5):688–706, 2006.