Domain Name System Anomaly Detection and Prevention

A Thesis Submitted in fulfillment
of the requirements for the degree of
Master of Science

by
Kiril Lashchiver

Supervised by
Prof. Danny Dolev
and
Shimrit Tzur-David

School of Engineering and Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel

September 2010

Acknowledgments

My sincere thanks to those who helped me with the whole deal.

Specially,
Prof. Danny Dolev, for his supervision and belief,
Shimrit Tzur, for the help and ideas,
my family, Marina and Yana, for love and understanding,
my wife Yulia Lovsky, for her time and support,
and of course Chappa and Kuza for being there in the hard times.

Abstract

The Domain Name System, or shortly - DNS, is a protocol that is used for a naming system for computers, services, or any other network resources. The most important purpose of the protocol is to translate names, which are meaningful to humans, into IP addresses, which are meaningful to computers, thus allowing us to easily locate every computer or service around the globe.

The DNS was invented in 1983 by Paul Mockapetris and his original specifications appear in RFC 882 and 883. These were later (November 1987) superseded by RFC 1034 and 1035. The earliest DNS servers were written in 1984 and 1985. The latest server was called BIND (Berkeley Internet Name Domain), it holds its name till today, with various versions.

DNS protocol was not designed to deal with security issues, therefore many hacking methods were developed in order to insert wrong data into the DNS lookup tables, or in its cache, every method that allows the insertion of wrong data into the DNS cache called 'cache poisoning' and most of the attacks on DNS servers attempt to perform that task. The main protection against those attacks in the DNS protocol is the Transaction ID (TID) of the request. Unless the reply is returned with the same TID as the original request, it is ignored. Therefore, in order to poison the cache of the DNS server, the hacker must guess the right TID for the request. In previous BIND versions, guessing the TID number was an easy task, since the TID field was increased by 1 for every new request. In the latest BIND versions, every new TID is a random number, thus making the guessing work much harder.

Nevertheless, there might be a situation when the hacker has an access to a randomly given TID field. Perhaps as a person that has control over the main routers, or he can in some way sniff the traffic of the network. In such a case the hacker can generate poison attacks without any difficulty. Our work concentrates in detecting and preventing such attacks. We assume that the hacker is an all knowing person and therefore knows the TID number of every request leaving the DNS server.

If an attack on the DNS Server successfully inserts invalid data into the DNS cache, it is called cache poisoning attack. Any query for the poisoned record will return the wrong IP to the user, thus directing him to the wrong site or service.

In this thesis we suggest algorithms to deal with such a case.

Contents

List of Figures

Thesis Overview

DNS protocol is one of the most popular and vital protocols over the Internet. The protocol is being used by us on a regular basis in order to translate names of sites or other network components into IP addresses. DNS is working in the background of our communication without us realizing when and how many requests we are sending. Due to its extensive usage, we naturally tend to assume that the replies returned by our local DNS server are both full and valid. This however, in not necessary the case. DNS servers, even those who are updated to the latest software release, can be hacked. Their records do not always contain the correct information and this false information might eventually be sent to the user, who will be redirected to a wrong site. The reason for that is the fundamental flaw in the lack of thought about security issues when the protocol was first designed, almost thirty years ago.

In this thesis, in Chapter 3, we present some of the work that was done over the years in order to prevent and minimize the attacks on the DNS servers. We explain the attacks and the solutions proposed to solve them. Many of the attacks presented, were either totally blocked or were minimized in order to lower the chance for a successful attack by new versions of the DNS servers. However, it is possible to assume that some attacks have not been found or have not been exposed to the public. Those zero-day attacks may be implemented and might have devastating effects on a command day.

All of the attacks presented assume that the hacker in not an all knowing person, meaning that the hacker cannot see the traffic in the network. Instead the hacker must rely on guessing, or carrying out massive searches. Both the guessing and the massive search attacks, might trigger alarms at network monitoring devices. In our work, we break this assumption and look on the case where the hacker can examine any traffic leaving or incoming the DNS servers. In this case the hacker can easily generate valid reply and send it to the DNS server, bypassing the network monitoring devices.

In Chapter 4 we describe our developed algorithm that identifies packets with anomalous time arrival in relation to other packets with same characteristics (i.e: replies of type A from certain Authoritative DNS Server). The packets with anomalous time arrival are treated as anomalous packets and are delayed for a certain period of time. Due to that, the algorithm is called 'Delay Fast Packets algorithm' or, in short, DFP algorithm. In case of a real attack, in the period of the delay a valid reply from the real Authoritative DNS server arrives and the DFP algorithm detects the case of multiple replies for one query and thus deduces that the DNS server is under attack. The DFP algorithm drops both replies and lets the DNS server issue a new query, as if regular packet loss had occurred.

We believe that the DFP algorithm, combined with other attack prevention and detection techniques already implemented on the DNS servers, will drastically lower the chances for a successful attack. Now not only the hacker needs to send the correct data, he also has to manage to fit the reply into an a period of time he can only guess.

At the end of the work in Chapter 5 and 6 we present in details the experiments we conducted on real traffic captured on HUJI DNS server and describe the strong and the weak points of the DFP algorithm.

Part I

DNS security

Chapter 1

Introduction

DNS protocol was designed to make our lives easier, as described in RFC 1034 [1] and RFC 1035 [2]. In the same way as we do not remember all the phone numbers in the phone book we do not remember all the IP addresses of all the sites. This is why DNS is very similar to the phone book service, the user gives it a name of the site, for example 'google.com', and it returns him '209.85.229.106', to which user's computer can then send the request for the page.

This role of the DNS puts it in a sensitive spot: the user must trust the DNS server to return the correct result for his request. If, for any reason, the DNS server makes a mistake, and returns the wrong IP address to the client, the user will access a different site while assuming he is accessing the real 'google.com'.

This mistake can occur because of the DNS caching system, which is used by the DNS servers for speeding up the process of the requests. Hackers search for opportunities to place faulty records into DNS's cache. Once the hacker managed to implant such a record (poison the cache) in the DNS, every user that requests this (poisoned) record will receive the malicious site.

There are two main ways to attack a DNS server. The first way is by finding bugs in the way the DNS protocol is implemented in a specific version of Berkeley Internet Name Domain (BIND) [3]. In order to prevent this type of attacks, BIND versions are constantly updated to fix the bugs in their DNS implementation. The second way to attack a DNS server is by finding flaws in the protocol. In our research we will concentrate on finding a solution to the second type of attacks. This way of approach allows us to find a more fundamental solution to the lack of security in DNS.

Potential consequences of DNS Cache Poisoning

- Identity Theft
  The hacker might create a site identical to the original site (for example 'paypal.com'). When the user connects the site, using the poisoned cache record, he might leave personal information like user name, password, phone number, address, etc. in the hands of the hacker.

- Distribution of Malicious Code
  One of the hacker's objectives is to distribute a malicious code. One way to achieve the goal is to release the code into the Internet and wait. This, however, will usually get random and few results. Poisoning the cache and impersonating some original and popular site will focus the attack. Once a user initiates a session with the hackers site the malicious code can be downloaded to the user's computer without his knowledge.

- Dissemination of False Information
  False information or false advertising can be used to spread self serving information about an organization. This false information can be considered as genuine when obtained from a trustworthy site like www.nasdaq.com.

- Man-in-the-middle Attack
  In this attack the user opens a session with the hacker's site, which in his turn opens a session with the originally requested site. The user continues his work with no suspicion, while all information that is flowing between the user and the genuine site passes through and is intercepted by the hacker's site.

DNS protocol flow

The following part will give a short explanation of the protocol, while emphasizing the important issues:

In the 'Name Resolution process' there are two types of queries that a client can make to a DNS server: recursive and iterative. While discussing 'name resolution', one DNS server can be a client to another DNS server.

- Recursive queries

  In a recursive query, the queried DNS server is required to respond with the requested data or with an error stating that the data of the requested type or

the domain name specified doesn't exist. The name server cannot just refer the client to a different name server. This type of query is typically done by a client to its Local DNS Server.

- Iterative queries

  In an iterative query, the queried DNS server returns the best answer it currently has back to the client. This type of query is typically done by a DNS server to other DNS servers after it has received a recursive query from a client.

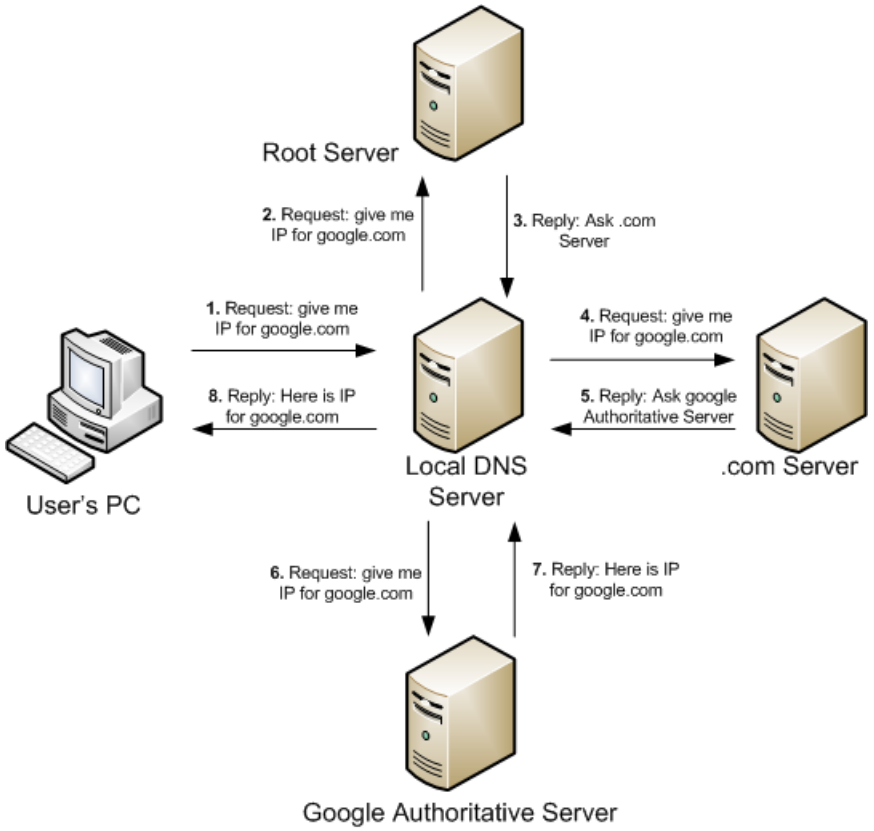

Figure 1.1: DNS protocol flow

Figure 1.1 shows an example for iterative query type, where all the queries are handled by the Local DNS server.

When an end user wants to access a certain site, for example google.com, his browser initializes the following process:

1. The local cache of the end user is checked. If an answer is found, the cached site is returned. If not, the end user sends query to the local DNS server.
2. The local DNS server receives the request and checks in its cache for an answer. If an answer is found, it is returned. If it is not found, the 'local DNS server' sends the request to the root server.
3. The 'ROOT DNS server' returns a redirect reply with the '.com DNS server' IP.
4. The 'local DNS server' sends the request to '.com DNS server' for the IP address.
5. The '.com DNS server' returns a redirect reply with the 'Authoritative DNS server' IP of the requested site.
6. The 'local DNS server' sends the request for the IP of the site to the 'Authoritative DNS server'.
7. 'The Authoritative server' finds the requested IP address for the given site name and sends the reply to the 'local DNS Server'.
8. The local DNS server returns the IP address and inserts the result to it's cache, in case anyone will ask for the same site in the near future.

DNS packet format

Figure 1.2 displays the format of a DNS packet, the important fields are explained in this section.

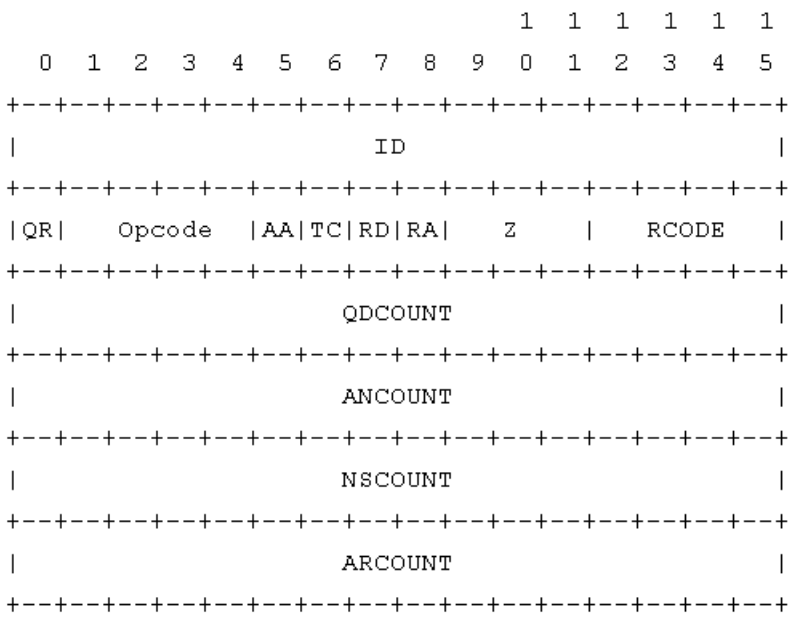

Figure 1.2: DNS packet format

- **ID** (or Transaction ID). 16 bits. Is used to match request and reply packets.
- **QR**. 1 bit. Request or Response flag.

QR	Description
0	Request
1	Response

- **AA**. 1 bit. Authoritative Answer indicates that the responding DNS server is the authority for the requested domain name in the corresponding request.

AA	Description
0	Not authoritative
1	Is authoritative

- **Rcode**. 4 bits. Return code. The following table describes the basic return codes as they were defined in RFC 1035.

Rcode	Description
0	No error.
1	Format error.
2	Server failure.
3	Name Error.
4	Not Implemented.
5	Refused.

- **Total Questions**. 16 bits.
  Number of entries in the request list that were returned.

- **Total Answer RRs**. 16 bits.
  Number of entries in the response resource record list that were returned.

- **Total Authority RRs**. 16 bits.
  Number of entries in the authority resource record list that were returned.

- **Total Additional RRs**. 16 bits.
  Number of entries in the additional resource record list that were returned.

- **Questions**[]. Variable length.
  A list of zero or more request record structures. Each request contains a query name, a query type and a query class. The query class is normally 1 for internet address. The most popular query types are listed in the table 1.1 below.

Type	Value	Description
A	1	Host's IP address
NS	2	Host's or domain's name server(s)
CNAME	5	Host's canonical name, host identified by an alias domain name
MX	15	Host's or domain's mail exchanger
ANY	255	Request for all records

Table 1.1: Popular DNS Query Types

- **Answer RRs**[]. Variable length.
  A list of zero or more response record structures.
- **Authority RRs**[]. Variable length.
  A list of zero or more authority record structures.
- **Additional RRs**[]. Variable length.
  A list of zero or more additional record structures.

Work overview

In this thesis we developed a new algorithm to detect anomalies in DNS packets. The algorithm identifies packets with anomalous time of arrival with respect to other packets with the same characteristics. Those packets considered as 'too fast' packets and are delayed for a certain period of time. In case of a real attack, in the period of the delay a valid reply from the real Authoritative DNS server arrives and the algorithm detects the case of multiple replies for one query and thus deduces that the DNS server is under attack. The algorithm drops both replies and lets the DNS server issue a new query, as if regular DNS packet loss occurred.

The algorithm was tested on real traffic at a HUJI DNS Server. We used a set of experiments to find the optimal parameters for the algorithm. The algorithm uses these parameters to detect these 'too fast' packets.

Chapter 2

Notations and Definitions

This chapter provides the terms and notations, which are commonly used in the IP/TCP based networks and DNS protocol specifications. These terms are necessary for a full understanding of the algorithm purposed in this work. Further information about IP, TCP and UDP can be found in RFC 791 [4], RFC 793 [5] and RFC 768 [6]

2.1 General Definitions

Definition 2.1 *Define* ***5 Tuple*** *as a five elements tuple representing a connection in an IP/TCP based network. The 5 Tuple contains IP source, IP destination, TCP source port, TCP destination port and IP next protocol (typically TCP or UDP).*

Definition 2.2 *Define* ***local DNS server*** *as the default DNS server for the user. Each DNS request, from the user, is first forwarded to the local DNS server, and only if the request can't be fulfilled by the local DNS server it is forwarded to other DNS servers.*

Definition 2.3 *Define* ***Authoritative DNS server*** *as the DNS server that gives answers that have been configured by an original source, in opposed to answers that were obtained via a regular DNS query to another Authoritative DNS server and were stored in the cache.*

Definition 2.4 *Define* ***DNS cache poisoning*** *or shortly* ***cache poisoning*** *as a maliciously created or unintended situation that provides data to a caching name*

server that did not originate from authoritative Domain Name System (DNS) sources. Once a DNS server has received such non-authentic data and caches it for future performance increase, it is considered poisoned, supplying the non-authentic data to the users of the server.

2.2 Round Trip Time

DEFINITION 2.5 *Define* ***Round Trip Time(RTT)*** *as the time required for a packet to be transmitted from a specific source to a specific destination and back again.*

The RTT usually depends on various factors including:

- The data transfer rate of the source's Internet connection.
- The medium used for the transmission (copper, optical fiber, wireless or satellite).
- The distance between the source and the destination.
- The number of machines connecting the source and the destination.
- The traffic load on the network.
- The number of other requests being handled by intermediate nodes and the remote server.
- The speed with which intermediate nodes and the remote server function.

The RTT can range from a few milliseconds, when calculated between close points (in same LAN), to several seconds, under worse conditions between points separated by a large distance (as in satellite connection) .

In order to estimate the quality and the latency of the connection, end point users constantly measure the RTT of their packets. However, due to the large number of variables and unknowns, especially in the internet environment, RTT estimation may vary drastically.

2.2.1 TCP

After the source transmits a Transmission Control Protocol (TCP) packet to its destination, it must wait a certain period of time for an acknowledgment. If the acknowledgment failed to arrive within the expected period of time, the packet is assumed to be lost and it is retransmitted. The question 'How long TCP must wait?' strictly connects to the expected RTT time of the network. Over local area connection (LAN), no more than a few microseconds will suffice. Over the Internet, a few seconds would be required. And if the traffic goes over satellite connections around the globe much more time is needed.

Since TCP must be able to work in every environment, TCP monitors the normal exchange of data packets and develops an estimation of how long it should wait for the next packet. This process is called 'Round Trip Time estimation'. Correct estimation has a crucial effect on the performance of the network: If the RTT is too low, many packets will be retransmitted unnecessarily, thus slowing the network. If the RTT is too high, the connection will spend valuable time sitting idle, waiting for the timeout to arrive.

Due to the fact that every network has different characteristics, TCP must be able to adjust the RTT estimation accordingly. Moreover, the characteristics, or the layout of the network can change rapidly both for better (higher speed) or for worse (lower speed). For example, switching from satellite medium communication to optic fiber medium makes the network faster, while changing from 'Data over Carry Pigeon' to 'Data over Snail' makes the network slower. Therefore, to be able to adjust to the changes in the network, the TCP new RTT estimation should be calculated for every new incoming TCP packet.

DEFINITION 2.6 *Define* ***EstimatedRTT*** *as the predicted RTT of the next packet sent by the source.*

The following formula shows how the new Estimated RTT is derived from the old Estimated RTT and the new RTT sample:

$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT' + \alpha \times RTT \ .$$

The default α in today's TCP implementation equals to 0.125. Such α means that in EstimatedRTT calculations more weight is given to the 'history part' than to the 'newly received part'. This is done in order to smooth any negative or positive

peaks. But even if a rapid change occurs in the network characteristics, TCP will adjust it's Estimated RTT in a very short time.

In order to identify lost packets TCP takes 'safety margins' as a deviation of the RTT from the Estimated RTT.

DEFINITION 2.7 *Define* ***DevRTT*** *as the deviation of the RTT from the EstimatedRTT.*

The following formula shows how the new DevRTT is calculated:

$$DevRTT = (1 - \beta) \times DevRTT' + \beta \times |RTT - EstimatedRTT| \ .$$

The default β in today's TCP implementation equals to 0.25. Then the Timeout is set to:

$$TimeOut = EstimatedRTT + 4 \times DevRTT \ .$$

If an acknowledgment for a packet arrives after $Timeout$ has elapsed (or does not arrive at all), the packet is considered to be lost and will be retransmitted by the source.

2.2.2 DNS

DNS mostly uses User Datagram Protocol (UDP) as a fourth lever protocol. DNS queries usually have relatively small payload, which fits into single UDP packet. Request from a client is followed by a single UDP packet from the server. Since UDP does not provide any packet loss detection, the DNS Server must be able to either issue a retransmission by itself or continue working without the lost packet. Of course, in the DNS case, the lost request will be retransmitted. The basic RFC advises that the retransmission interval should be based on prior statistics, if possible. Too aggressive retransmission can easily slow responses for the community in large networks. The minimum retransmission interval depends on how well the client is connected to its expected servers. Usually it should be 2-5 seconds.

Chapter 3

Related Work

3.1 Attacks Overview

DNS is one of the most important services both in private networks and over the Internet. It is hard to imagine how, without its service, we would surf the net, or use any of its applications.

However, DNS popularity and our dependence on it, is also one of its drawbacks. Every new connection to a site or to an application, usually begins with DNS query for the IP address of the destination. If for any reason the returned IP is incorrect the user might end up in hackers hands.

The main reason for such a problem is that the DNS protocol lacks any reference or design of security. Therefore, over the years, the DNS Servers were constant target of hackers. In the meanwhile the academy and the security companies made every effort to prevent the attacks by fixing and patching the DNS Servers (thus improving the software that implements the protocol). However, no changes were inserted into the protocol itself.

This section will cover some of the most popular attacks over the years and what was done to prevent them. And although some of the following attacks are no longer possible, they are still interesting, since they emphasize the fact that DNS was not designed as a secure protocol, thus DNS implementation was initially flawed. The security limitation developed over the years whenever attacks were discovered. Hence it is possible to assume that some of the attacks, on the most updated BIND version are still unknown or not discovered. More fundamental work must be done to prevent Zero-Day attacks from succeeding. Hence, as was mentioned, this work

suggests an algorithm to detect and prevent Zero-Day attacks.

We will start with a presentation of some of the common DNS attacks as they were published during the years in the Internet. All of the attacks are widely known, but some still present a threat to even the most updated DNS server.

3.1.1 Additional Data Attack

The idea behind this attack is to issue a valid request and generate a valid reply followed by some additional data. This data will be welcomed into the cache of the DNS Server for future use. The following two examples for attacks are quite old and will not work on the new BIND versions, but it is worth mentioning them, for better understanding of the cache poisoning attempts and their potential damage. The user can test his DNS server with the following website [7] for this vulnerability. Figure 3.1 presents the packets flow of the two attacks.

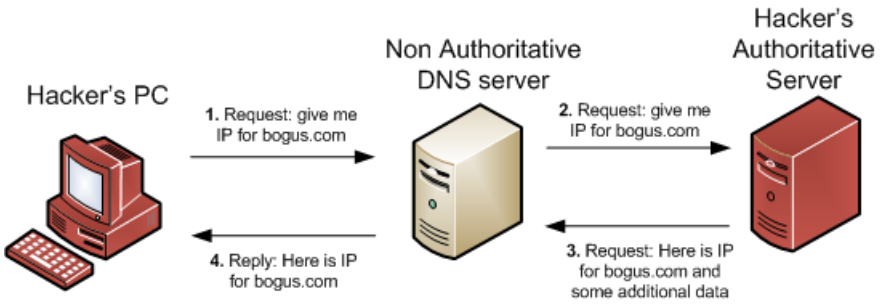

Figure 3.1: Additional data attack

Unrelated Data Attack

The hacker issues a request to the DNS Server for some nonexistent domain, from his controlled Domain. Since the DNS has no record for that domain, it will issue a request to the Authority DNS Server of the domain. After receiving the request, the hacker sends a valid reply, while adding an additional data. The additional data contains IP address for domains that were not included in the original request. The IPs of the "nonexistent" domain and the added (not requested) domains are stored in the cache of the DNS Server for future use.

This attack does not work anymore: No domain is cached except of the original domain in the request.

Related Data Attack

The hacker uses the same technique as in 'unrelated data attack', but this time the data is some extra information that is related to the queried domain. Usually the extra data contains MX, CNAME and NS records, those records point to the data the hacker wants to be cached.

This attack does not work anymore: No data except of the queried data allowed into the cache.

3.1.2 Transaction ID Guessing

The only real obstacle that stands between the hacker and successful cache poisoning of the DNS Server is the Transaction ID field in the DNS protocol. Therefore, the hackers look for weak spots in the protocol implementation that would allow them to make a good guess of the Transaction ID and this way interfere with the traffic. The next section is dealing with methods used by the hacker to overcome this obstacle and with how the DNS Servers can be defended against it. Figure 3.2 presents the packets flow of the attack.

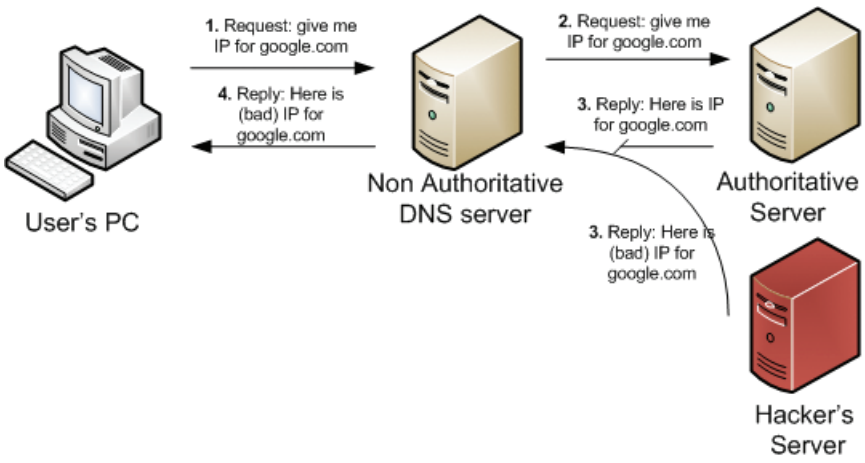

Figure 3.2: Transaction ID Guessing

Increment by One

The earliest BIND servers did very little for security issues. In order to avoid same Transaction ID repeating at the same time in the network, the Server used "Increment by One" method. Each new query was issued with the previous $TransactionID+1$. Guessing the Transaction ID in such a case is a fairly easy job. This weakness was patched and the new BIND versions issue random Transaction ID to each new query.

Deriving Transaction ID number in BIND 9

As mentioned above, the Transaction ID number could be easily guessed, in earlier BIND versions. However in the new (BIND 9) version the Transaction ID is a randomly generated number, or more precisely, the Transaction ID is a PRNG - Pseudo Random Generated Number. The algorithm that generates the PRNGs in each of the BIND versions is open to the public and can be easily obtained and studied. As shown in [8] BIND 9 versions 9.4.0-9.4.1, 9.3.0-9.3.4, 9.2.0-9.2.8, 9.1.0-9.1.3 and 9.0.0-9.0.1. the PRNG algorithm is weak and the next random number can be derived from the previous one. This particular problem was fixed in the 9.5.0 BIND version.

Birthday Paradox Attack

In order to perform this attack the hacker first sends, simultaneously, large quantity of packets to the DNS server, requesting the same Domain Name. The DNS server generates the same number of queries and sends them to the Authority Server. The hacker generates the same amount of DNS bogus replies with random Transaction ID. The birthday paradox dictates that a few hundreds of packets will suffice to promise 50% success to match between Transaction IDs of at least one query and one bogus reply. And thus manage in poisoning the cache of the DNS server. This attack was fully described in [9] and [10].

The Birthday attack guaranties high chances of success with relatively low number of packets required. While in regular packet spoofing the hacker sends N replies for one query, thus the probability of success is $\frac{N}{T}$ where N is the number of packets sent and T is the total number of packets possible (In the DNS case $T = 2^{16} - 1 = 65535$). In Birthday Paradox attack the hacker only needs to match

one of the requests to one of the replies, the probability of success can be calculated by the following formula:

$$P(success) = 1 - 1(1 - \frac{1}{T})(1 - \frac{2}{T})...(1 - \frac{N-1}{T}) = 1 - \frac{T!}{T^N(T-N)!};.$$

The power of 'Birthday Paradox attack' over the 'regular packet spoofing attack' is that it requires relatively small number of packets in order to make a successful attack. A mere 300 packets will promise 50% of success while 750 packets will promise 99% of success. In the regular packet spoofing attack with 750 of packets will only promise $\frac{750}{65535} = 1.14\%$ of success.

Birthday Paradox attack shows that even a randomly generated Transaction ID used in the latest BIND versions is vulnerable to brute-force attacks.

3.1.3 Flood Attacks

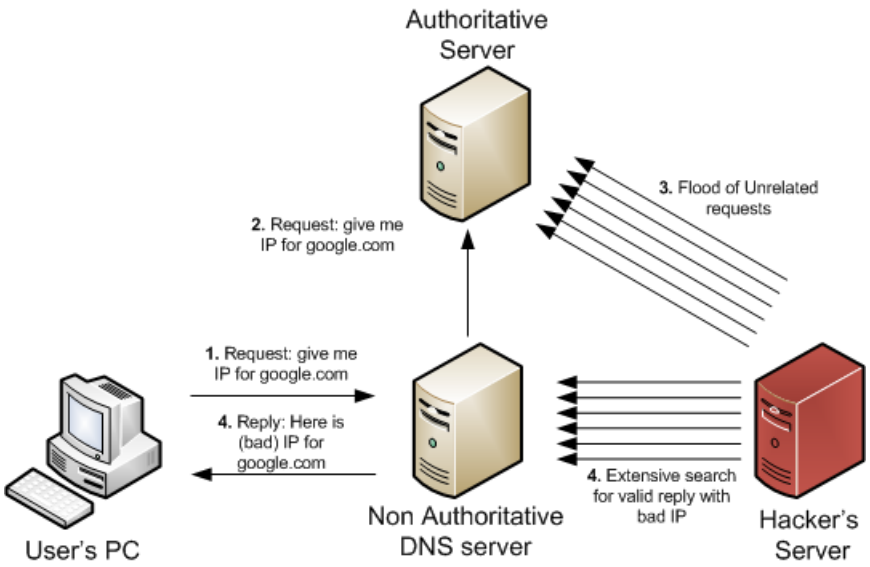

Figure 3.3: Flood attack

In order to succeed in a brute force attack the hacker needs as much time as possible. The time he can use starts when the DNS server generates the request and elapses when the answer is received from the Authority DNS Server. Usually it is not enough

to run extensive search for the valid Transaction ID on the packets. In order to gain more time, the hacker may try to flood the Authority DNS Server trying to slow it down and perhaps crashing it. The attack flow is similar to the Transaction ID Guessing attack and its flow presented in Figure 3.3

3.2 Solutions

There are several offered solutions on how to deal and prevent cache poisoning attacks and attempts [11], [12], [13], [16]. In this section we present some of them:

3.2.1 BIND Versions

BIND is the most used DNS software over the Internet [3], and because of that it is a constant target to hackers attacks. New versions and version updates are released constantly with new updates and patches for bugs and security issues. Therefore one of the most important ways one can enhance the security of your local DNS server is to run the most recent version of BIND, since all versions of BIND before the most recent one, are probably susceptible to at least a few known attacks [8].

However, new attacks are probably not yet patched by the new version, therefore can still affect your system. One way to find out about new attacks and the ways to protect against them is to read the comp.protocols.dns.bind newsgroup regularly [12].

3.2.2 DNS Configuration and Administration

One of the most popular mistakes is improper configuration of the DNS. When misconfigured even the most updated DNS server is vulnerable to many known attacks (not necessary cache poisoning) as described in [13], [14] and [15]. For example, one of the important configurations is the restriction of Zone Transfers commands to known and trusted IPs only.

When you are positive that the local DNS was attacked, there are several steps that can be made to prevent further damage:

1. Make sure you are not infected with a spyware. Many spyware programs modify the DNS configuration.

2. Try to find out the IP address of the malicious DNS server.

3. Insert the malicious IP address into IPs Black List.

4. Clear the cache of the DNS Server, either with resetting the server or with any other clear cache commands.

Other good practices when dealing with configuration are physically securing the DNS Servers, hiding the version of the running BIND [17], removing all unnecessary services running on the DNS Server, randomizing the UDP source port [18] and regularly monitoring the DNS Servers and DNS log files [21].

3.2.3 Question Modifications

The main assumption in the following solutions is that the DNS protocol will not change any time soon, hence some solution for the integrity of the transactions must be found in the bounds of current specifications. The basic idea is to find fields that can be modified in some way, and this modification will increase the number of queries to such an amount that a hacker will have no chance to find and send the right modification of the response before the real one from the Authentication server arrives.

1. Bit 0x20 [19]: The idea is to use the Question section to add random bits to the query, the DNS server will have to match those bits in any legitimate response it generates. Most of DNS servers do not care if the question is presented in upper or lower case, therefore a combination of the cases can provide the essential random bits to the query. The difference between 'A' (0x41) and 'a' (0x61) or 'Z' (0x5a) and 'z' (0x7a) is the 0x20 bit (hence the name of the algorithm). For example:

   WWW.IETF.ORG 000 0000 000
   WwW.iEtF.oRg 010 1010 101
   wWw.IeTf.OrG 101 0101 010
   www.ietf.org 111 1111 111

   The number of bits generated by this method is a function of the length of the domain, long domains will have many random bits, while short domains will remain with only few bits, leaving their sites still vulnerable to the above attacks.

2. Random prefix [20]: This method proposes to use Wildcard Domain Names to increase the entropy, for example if a user wants to resolve 'www.example.com, the DNS server will generate a random prefix for the query and send 'ra1bc3twqj.www.example.com A'. The Authoritative DNS server will return the same domain name with 'www.example.com' IP address. This method using prefix of length 10 will generate about $\log_2 36^{10} \approx 52$ bits.

3.2.4 Secure DNS

Since DNS protocol does not include any security, Domain Name System Security Extensions (DNSSEC) were developed as described in RFC 3833 [16], [22] DNSSEC was designed to prevent cache poisoning by having all its answers digitally signed, so the correctness and the completeness of the data could be easily verified.

DNSSEC is a new protocol and only lately some of its critical pieces were formally defined. Deploying the protocol on large-scale networks is a challenging task. The reason is that a minimal level of deployment is required for a user to gain benefit from the new service. It is hard to find the first users who, for a certain period of time, will be ready to pay the price of using a new protocol.

3.2.5 Our Contribution

In this work we propose an algorithm that deals with a stronger type of a hacker than the one assumed in previous solutions (excluding the DNSSEC). Our hacker can inspect (but not modify) any packet sent or received by the local DNS server. Our solution can be implemented as a black box just after the DNS server, therefore there are no modifications requirements nor in the DNS protocol nor in BIND server code.

Chapter 4

Algorithm: Delay Fast Packets

As was previously mentioned, DNS protocol usually uses User Datagram Protocol (UDP) as a forth level protocol for its data communication. If, for some reason, the request or the reply fail to reach their destination the DNS Server simply issues another request. In such a case, the DNS Server needs to know how to handle the situation that arises from packet delays, as these may be accidently interpreted as packet losses. The way DNS operates is the most straight forward way: simply accept the first valid reply (valid reply is a reply from Authoritative Server with correct Transaction ID) and ignore all other replies. In other words, only the first reply will be forwarded to the user and cached for future usage. This is a drawback in the DNS security and a gateway to hackers to attack the DNS and poison the cache.

This chapter presents a 'Delay Fast Packets algorithm' or shortly, DFP algorithm that detects and prevents attempts of cache poisoning attack. The idea for the DFP algorithm is the following observation: in order for an 'attempted attack' to become a 'successful attack' it must beat the real reply (from Authoritative Server) in a race back to the local DNS server. For succeeding in that, the hacker tries to send a reply as soon as possible, usually much faster than it takes to the Authoritative Server to generate a response. Our DFP algorithm identifies that exact point and therefore, detects and prevents the attacks.

4.1 DFP Algorithm Presentation

The main idea of the algorithm is to estimate the RTT between the DNS Server and each of the Authoritative Servers it ever encounters and delay the replies that arrive too fast according to the approximation (therefore it is called the DFP algorithm - Delay Fast Packets algorithm). Moreover, the DFP algorithm estimates the RTT for each service type the Authoritative Server can provide (MX, A, AAAA, CNAME, PTR etc...). For some of those services RTT estimation is higher than for the others due to more extensive Data Base search on the Authoritative side. For each Authoritative Server and service type, the Estimated RTT predicts the average time needed for the next reply to arrive. If for any reason, a reply comes too soon, according to the DFP algorithm, the DNS Server waits for a certain amount of time. If another valid reply arrives in that window of time, both replies are dropped, and a new request is generated (as if regular DNS packet loss occurred). If the attacker is persistent and sends reply for each request the user will experience Denial of Service attack since the DFP algorithm will not pass any of the replies back to the user. This attack is called DoS (Denial of service) and this subject is not covered in this work.

In the following section (4.2) we present a simplified pseudocode that demonstrates the idea of the DFP algorithm. In order to pass the idea without complicating the pseudocode, the following DFP algorithm does not take care of multiple packets attack. If the hacker decides to send multiple 'valid' replies for one request, the following DFP algorithm will only drop the first two and will fail in dropping the following hacker's replies. However, the generalization of the DFP algorithm is easy: for each fast packet, every reply for the same request must be saved for a certain amount of time, and then all the replies must be dropped. Of course in order to prevent an overflow in the memory of the DNS servers, only the minimal relevant information must be stored, like 5-tuple and time until the replies are no longer valid (the regular DNS timeout).

Another un-handled issue in the DFP algorithm is that 'too fast' packets affect the RTT estimations. In case where there are no attacks, those 'too fast' packets can mark a change in the topology of the network and thus must be considered in the RTT estimations. However, in case of possible attacks, it would be a bad idea to include them in the estimations, as they may be an attempt of the hacker to lower our RTT estimations in order to make a successful attack in the close future. Hence 'too fast' packets must not affect the RTT until we are sure they are real and valid replies and not attack attempts. This will be known only when a certain amount of

time elapses and no other replies arrive in that period of time.

4.2 Pseudocode

Algorithm 1 DFP - Delay Fast Packets

Main Method

```
1: PacketDictionary.Init()
2: StatsDictionary.Init()
3: loop
4:    NewPacket ⇐ SniffDNSPacket()
5:    ID ⇐ NewPacket.TransactionID + NewPacket.Type
6:    5Tuple ⇐ NewPacket.5Tupple
7:    IsQuery ⇐ NewPacket.IsQuery
8:    if IsQuery then
9:       if [5Tuple, ID] already in PacketDictionary then
10:         PacketDictionaly[5Tuple, ID] = NewPacket
11:         print DUP REQUEST FOUND: REMOVING PACKET AFTER TIMEOUT
12:      else
13:         PacketDictionaly.Insert([5Tuple, ID],NewPacket)
14:      end if
15:   else
16:      RequestPacket ⇐ PacketDictionary.Get([5Tuple, ID])
17:      if [5Tuple, ID] is delayed then
18:         Drop all packets with ID [5Tuple, ID]
19:         SEND: WARNING, 'TOO FAST' PACKET RECEIVED, POSSIBLE ATTACK
20:      else
21:         if RequestPacket not exists then
22:            SEND: WARNING, UNMATCHED REPLY FOUND, POSSIBLE ATTACK
23:         end if
24:         RTT ⇐ NewPacket.TimeOfArrival−RequestPacket.TimeOfArrival
25:         DelayTime ⇐ StatsDictionary[[NewPacket.SourceIP,
            NewPacket.ReplyType].AddSample(RTT,NewPacket.SourceIP,
            NewPacket.ReplyType])
26:         DelayPacket(DelayTime)
27:         PacketDictionary.Remove([5Tuple, ID)]
28:      end if
29:   end if
30: end loop
```

AddSample(RTT, IP, Type)

1: **if** First Sample for this IP and TYPE **then**
2: $EstimatedRTT \Leftarrow RTT$
3: $DevRTT \Leftarrow 0$
4: **end if**
5: $EstimatedRTT \Leftarrow (1 - \alpha) \times EstimatedRTT + \alpha \times RTT$
6: $DevRTT \Leftarrow (1 - \beta) \times DevRTT + \beta \times |RTT - EstimatedRTT|$
7: **if** $RTT < EstimatedRTT - DevRTT \times FactorWindow$ **then**
8: **return** $(EstimatedRTT + DevRTT \times FactorWindow) - RTT$
9: **else**
10: **return** 0
11: **end if**

4.2.1 DFP Algorithm Documentation

Main Method (detailed by code rows):

(1,2) : The DFP algorithm uses two hashtables. PacketDictionary for matching between the outgoing requests and the incoming responses. StatsDictionary for storing the statistics for each Authoritative DNS server.

(3) : Main loop, runs indefinitely, as long as the program is operating.

(4) : Receiving new packet from the network device.

(5,6,7) : Extracting the parameters from the new packet, getting the 5 Tuple, the Transaction ID, Packet Type and the IsQuery flag.

(8) : Management of the request packets.

(9) : Checking whether there is a packet with the same 5 Tuple and ID in the PacketDictionary hashtable.

(10,11) : Replacing the new request with the old request in the PacketDictionary hashtable. Note: This condition can only happens if no reply was received and a new issued request has same 5 Tuple and the same ID as the old request.

(12,13) : Inserting the new request into the PacketDictionary hashtable.

(15) : Management of the reply packets.

(16) : Getting the matching request packet with the right 5 Tuple and ID.

(17) : Checking that no delayed replie has same ID as the current reply.

(18,19) : If multiples IDs found, drop all of them and sends a warning.

(21,22) : If no matching request is found (we've got a reply with no request), warning is sent, as this condition can never be fulfilled unless there is an attack on the DNS server.

(23) : Management of the normal case when both reply and request are found.

(24) : Calculating the RTT between the DNS servers by measuring the differences between the time of arrival of the request and the reply.

(25) : Running AddSample method on the new sample and the RTT.

(26) : Preventing the reply packet from entering the DNS server core, for the amount of time indicated by the AddSample method.

(27) : Removing the request from the PacketDictionary Hashtable.

AddSample:

(1) : Checking whether this is the first sample for the IP and Type.

(2,3,4) : Initializing the Statistics parameters.

(5) : Calculating the EstimatedRTT with the given α.

(6) : Calculating the DevRTT with the given β.

(7) : Checking whether the packet is too fast according to the window determined by the EstimatedRTT, DevRTT, FactorWindow parameters.

(8) : If it is a fast packet, return the time interval for which it must be delayed.

(9,10,11) : Otherwise, return 0.

4.3 DFP Algorithm Specifications

The DFP algorithm uses the following formula in order to detect 'too fast' packets: $RTT < EstimatedRTT - DevRTT \times FactorWindow$. Each DNS reply that arrives too soon according to the formula is considered suspicious and delayed, giving enough time for a possibly other reply with the same TID to arrive. The variables in the formula are controlled by three parameters: α, β and $FactorWindow$. The performance of the DFP algorithm, in terms of speed, detection accuracy and memory allocation, depends on how well those parameters are configured. In this section we describe the consideration and the experiments that led us to choose the values for these three parameters.

4.3.1 Window

Definition 4.1 *Define* ***Window*** *as a certain point in time beginning after issuing the DNS request, each reply that arrives before the window begins is considered suspicious, i.e: for www.xkcd.com Authoritative DNS server with type A, the window might begin at 3400 ms, while for type MX it might begin at 3800.*

Definition 4.2 *Define* ***Window Starting Point*** *as the elapsed time since the request to the beginning of the window.*

Definition 4.3 *Define* ***False Alarm*** *as a packet originated by the Authoritative Server and arrives before the Window starting point and even if enough time is passed no other reply with same TID will arrive. Each of those false alarms makes the DNS server store the packet in memory for a short time and release it only after it is safe.*

The window is of a very dynamic nature, its starting point constantly changes and shifts on the time axis. This is due to the dynamic nature of the Internet network and the constant changes in the RTT of the arriving requests.

The window starting point dictates which packets are considered 'too fast' and thus needed to be delayed, and which packets are in the normal time boundary thus can pass through without any delay. In order to adjust the parameters that define the window starting point, there are two main points to consider:

- Early starting point allows more packets to pass through, without delay. On the one hand, this helps the DNS server to work faster without delaying too

many suspected packets (until it is safe to pass them on). However, on the other hand, carefully planned attacks can try to hit just above the starting point thus pass the filter without triggering an alarm.

- Late starting point delays more packets since it considers them as 'too fast' packets. On the one hand, this makes the work of the hacker much more difficult, making him compete on much smaller interval of time and thus making his cache poisoning attempt much harder. However, on the other hand, late starting point forces the DNS server to delay many packets, considering them as a potential threat, this results in slower responses of the DNS server to the users and much more memory consumption inside of the DNS server.

In the following sections (4.3.2 and 4.3.3) we refer to α, β and the $FactorWindow$. We perform a set of experiments in order to demonstrate the influences of each of the parameters on the window starting point and hence on the tradeoff between the number of false alarms and the probability of detecting and preventing a protectional attack. Please note that in order to emphasize the results more clearly the two sets of experiments were made on different data sets (while the conclusions are the same on each of the data sets available to us, we tried to find as clear graphs as possible).

4.3.2 α and β Considerations

α and β are the two parameters influencing the EstimatedRTT and the DevRTT parameters in the DFP algorithm. They determine the weight of the new RTT sample against the history, thus, influence the window starting point. In order to find how α and β influence the number of 'fast packets' detected by the DFP algorithm, we conducted several experiments on real traffic without any attempted attacks. In each experiment the number of false positives alarms was determined. The following figures 4.1, 4.2, 4.3 demonstrate the results of the experiments run on exactly the same traffic, using different values of α and β each time.

Note: The $FactorWindow$ parameter is set to 2 in each of the following experiments.

Figure 4.1, below, deals with the case of low values of α and β:

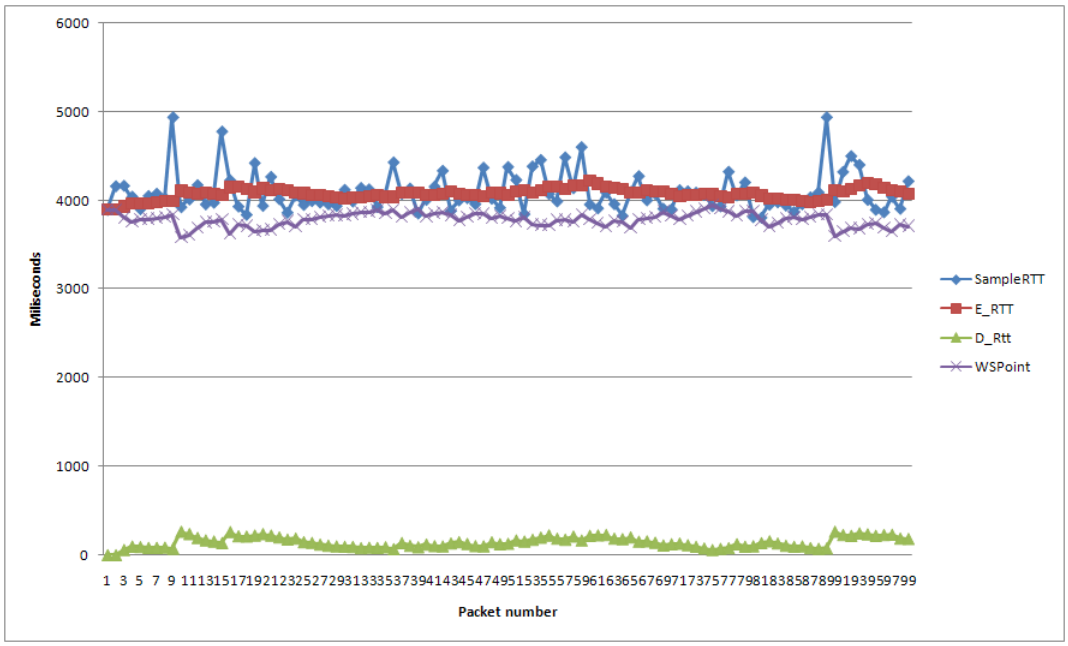

Figure 4.1: α=0.125, β=0.25, $\#FastPackets = 2$

Low α and β values, as in TCP RTT estimation, smooth the Estimated RTT function, since more weight is given to the history of the samples rather than to the newest sample in comparison to higher values of α and β in the following experiments. In each of the experiments the newest sample is given a much higher weight, since it is better in predicting the future RTT. Estimated RTT is represented by a red curve in the graph. However, for each peak in the RTT (dark blue curve) the DevRTT (green curve) rises, as expected from the window starting point equation. (Note, for example, the peak in RTT and rise of DevRTT at packet number 10). As an immediate result, the window starting points lowers. Window starting point is shown in purple color. This situation allows potentially bad packets a wider window of opportunity to attack the DNS server.

We see that only two packets were considered 'too fast' in this configuration: packet number 40 and packet number 79. It is shown in the graph that those packets RTT time is exceeded the starting point.

Figure 4.2, below, deals with the case of high values of α and β:

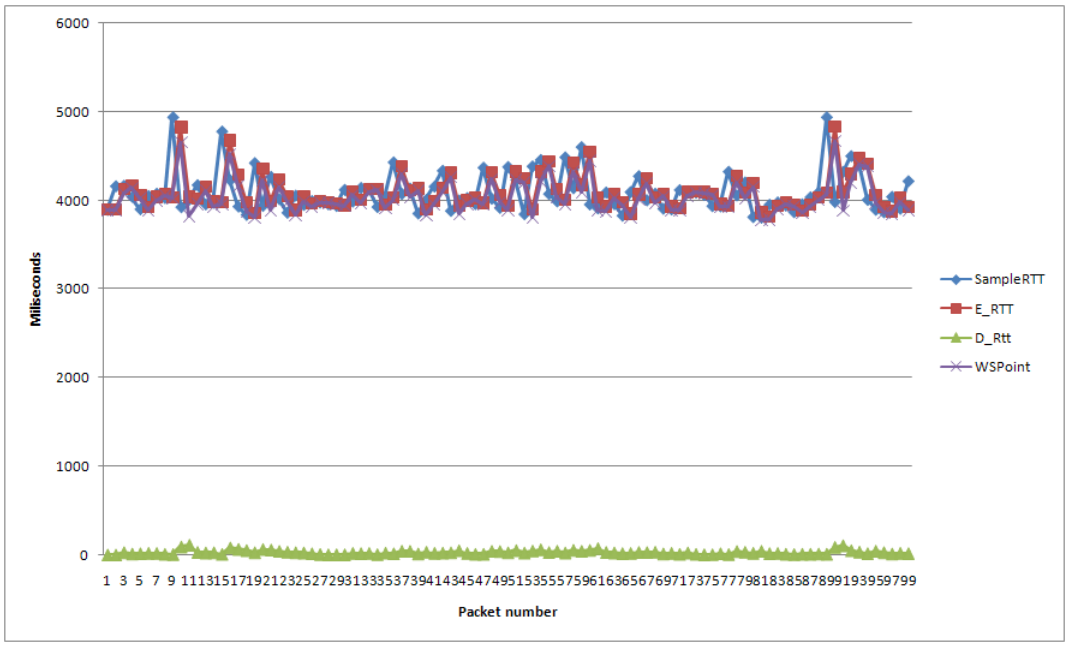

Figure 4.2: α=0.875, β=0.75, $\#FastPackets = 23$

High values of α and β, give most of the weight to the newest sample (in comparison to other experiments). Hence the RTT deviation (green curve) is very small, the window starting point is extremely late (purple curve) making DNS attacks attempts very hard to succeed. However, this situation also creates many false alarms, as any fluctuation in the RTT will probably put the new sample before the window starting point.

We see that in this configuration about one fifth of the packets were considered 'too fast'.

Figure 4.3, below, deals with the case of medium values of α and β:

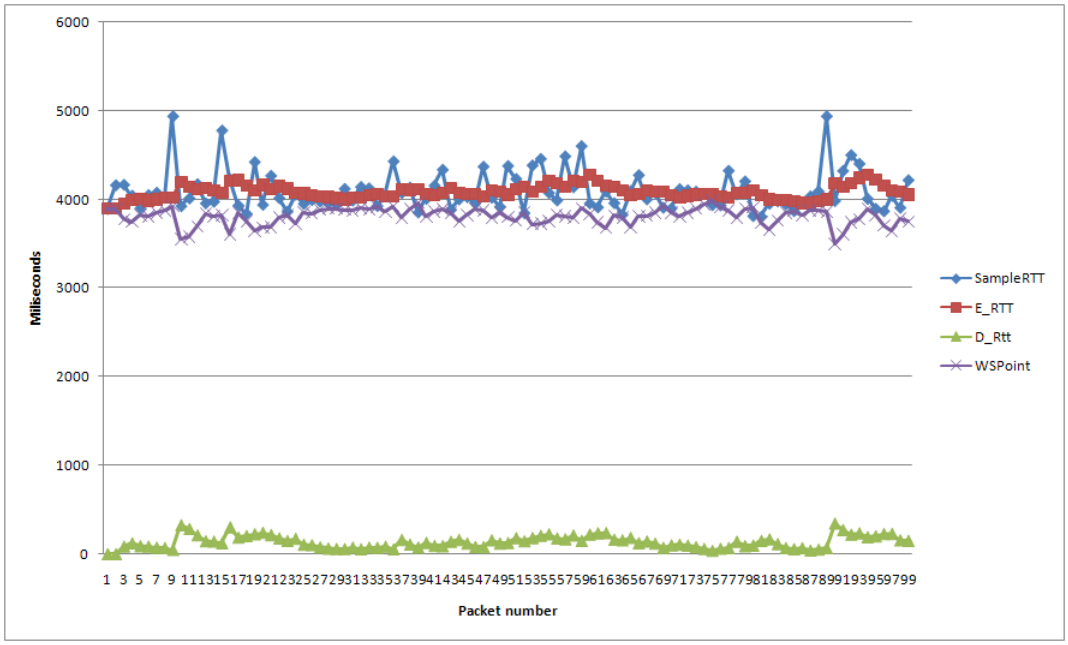

Figure 4.3: α=0.2, β=0.4, $\#FastPackets = 5$

The values of α and β, are the median of the 'Low' and 'High' configurations. The starting point that is created in this case is later than in the 'Low' configuration and the RTT deviation (green curve) is higher than in 'High' configuration case.

We see that in this configuration five of the packets were considered 'too fast'.

Our experiments show that most of the time the deviation of the RTT is relatively low, therefore the created starting point is rather high. The change in the deviation occurs when an extremely slow packet arrives, in this situation the window starting point is lower for a short period of time and possible attacks have higher chances of success. However, our experiments show that the slow packet situation occurs seldom and we think it is more important to prevent false alarms that might be created by valid 'too fast' packets. This consideration brought us to the decision that the chosen configuration in our implementation of the DFP algorithm is the 'Low' version. However if the local DNS server can afford to maintain larger memory

consumptions created by the false alarms, the better configuration is the one that prevents more attacks, and in that case it is better to choose the 'Medium' or even (if memory is not a problem) the 'High' configuration.

4.3.3 FactorWindow Considerations

After setting up the α and β parameters the configuration of the $FactorWindow$ parameter should be determined. This parameter goal is to lower the starting point created by α and β, by a constant factor. As before, the tradeoff between the number of false alarms and the probability of a successful attack dictates which value will be chosen.

Note: The α and β parameters are set to 0.125 and 0.25 respectively in each of the following experiments.

Figure 4.4, below, deals with the case where $FactorWindow = 1$:

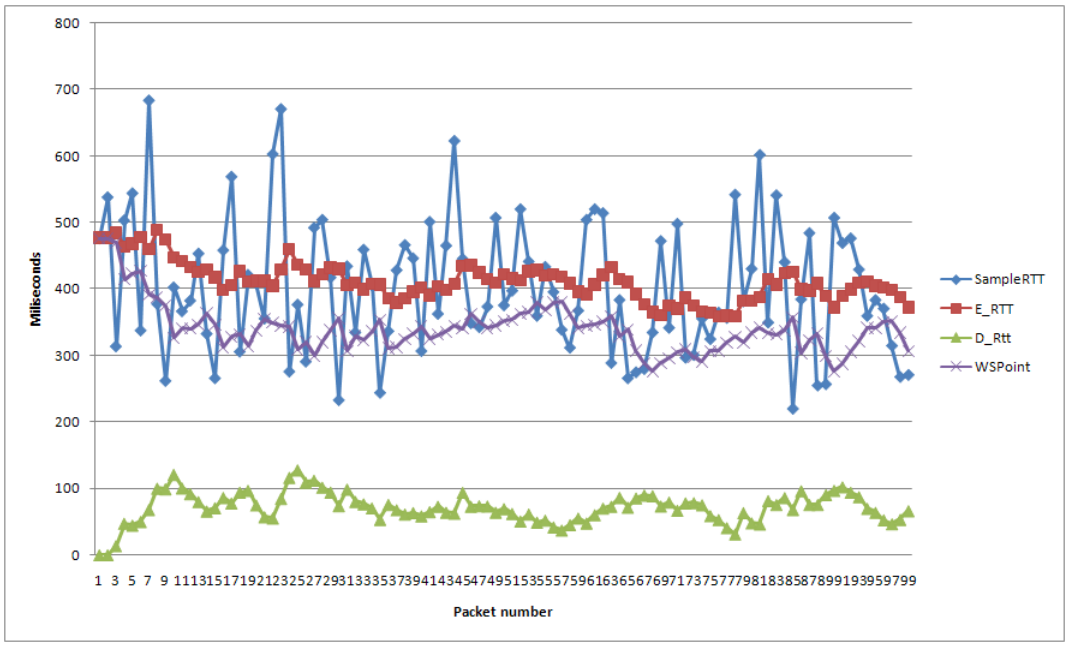

Figure 4.4: Factor Window value 1, $\#FastPackets = 28$

$FactorWindow = 1$ means that the window starting point is modified only by α and β, therefore the created starting point is high and the probability for a packet to come before the starting point is high. In this case many packets have to be delayed.

We see that in this configuration, about one third of the packets are considered to be 'too fast'.

Figure 4.5, below, deals with the case where $FactorWindow = 2$:

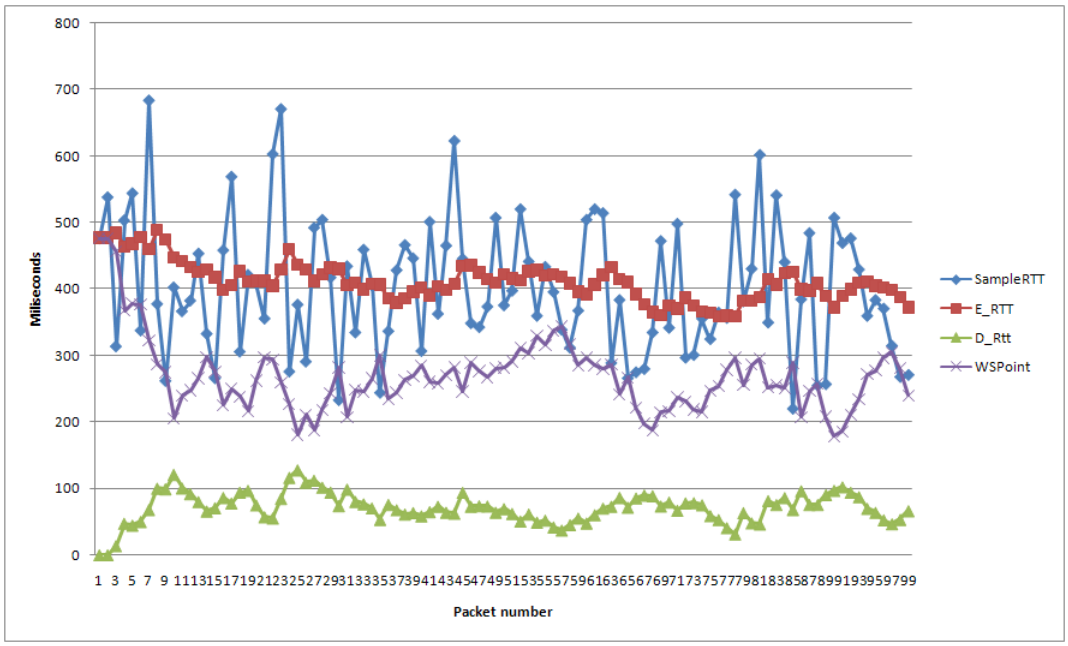

Figure 4.5: Factor Window value 2, $\#FastPackets = 9$

$FactorWindow = 2$ means that the starting point is two estimated deviations from the estimated RTT. Using 'Chebyshev inequality' the probability for a packet to exceed the starting point is less than $\frac{1}{4}$. However, in practice, the bound is tighter. Our experiments show that only about $\frac{1}{10}$ of the packets are considered as 'too fast'.

We see that in this configuration, nine of the packets were considered 'too fast'.

Figure 4.6, below, deals with the case where $FactorWindow = 3$:

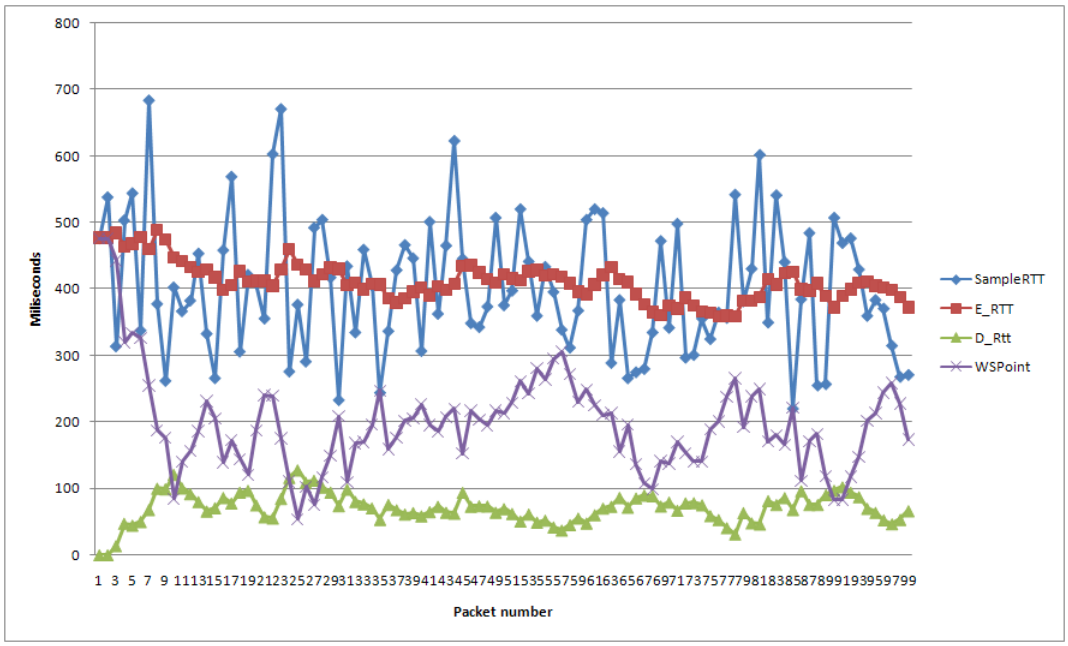

Figure 4.6: Factor Window value 3, $\#FastPackets = 1$

$FactorWindow = 3$ means that the starting point is three estimated deviations from the estimated RTT. Using 'Chebyshev inequality' the probability for a packet to exceed the starting point is less than $\frac{1}{9}$. However, in practice, almost no packet exceeds the starting point, it is so low that the attacker can easily infiltrate, even without knowing that there is a detection and prevention DFP algorithm running.

We see that in this configuration, only one of the packets is considered 'too fast'.

The main consideration for choosing the configuration of the $FactorWindow$ parameter is again the tradeoff between the number of false positives and the probability of successful attack. Using the results from the experiments, the value of $FactorWindow$ should be between 2 and 3. We have decided that the DFP algorithm will run with $FactorWindow = 2$.

4.4 Weaknesses

DEFINITION 4.4 *Define* ***Window Ending Point*** *as a point above the Estimated RTT. Each reply arriving after the ending point will be considerer as 'too slow' packet.*

The main assumption of the DFP algorithm is that the deviation from the EstimatedRTT is close to zero. As it will be shown in the 'Simulation' part, this assumption was proven to be true in many experiments carried out on real traffic. But in some cases, the deviation rises for short periods of time. In those moments, the opportunity for potential attacks is opened, since many 'too fast' packets fall into the window starting point of : $(EstimatedRTT - DevRTT) \times FactorWindow$ bound.

This situation occurs after a very slow packet is received. The 'too slow' packet creates temporary increment of the deviation and lowers the starting point, as seen in figures 4.7 and 4.8. The starting point returns to normal parameters after about 3-5 packets when the influence of the historic 'too slow' packet weakens. (The influence depends on the α and β configuration used in the DFP algorithm, as explained in previous section.) The temporary lowering of the starting point creates an opportunity for a hacker to attack the DNS server.

Example 1:

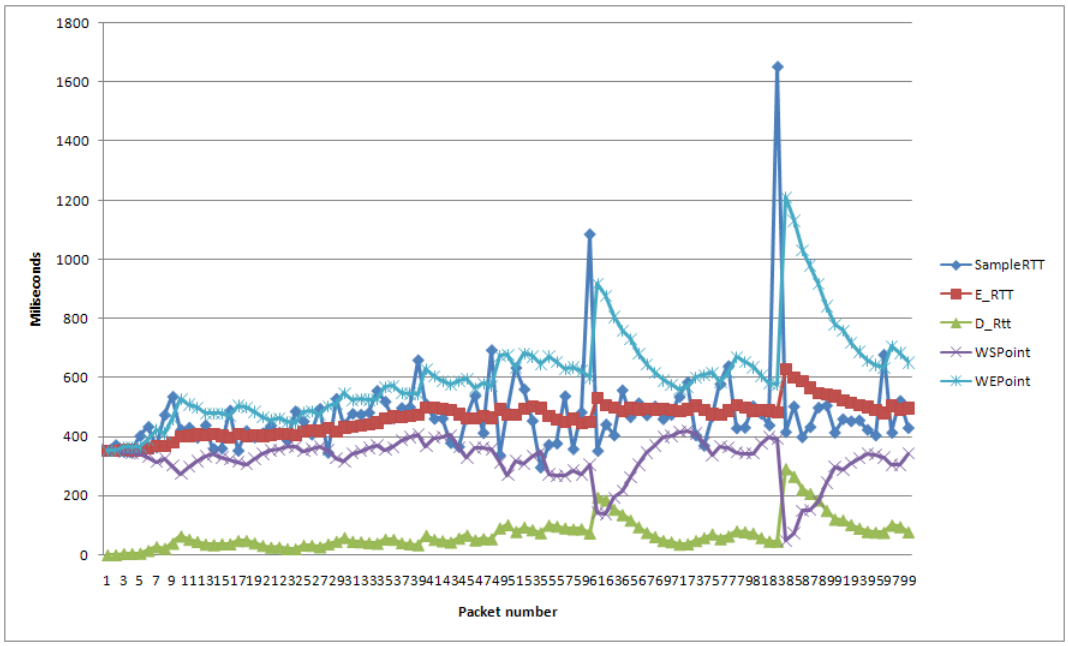

Figure 4.7: Low starting point created by high peaks

Example 2:

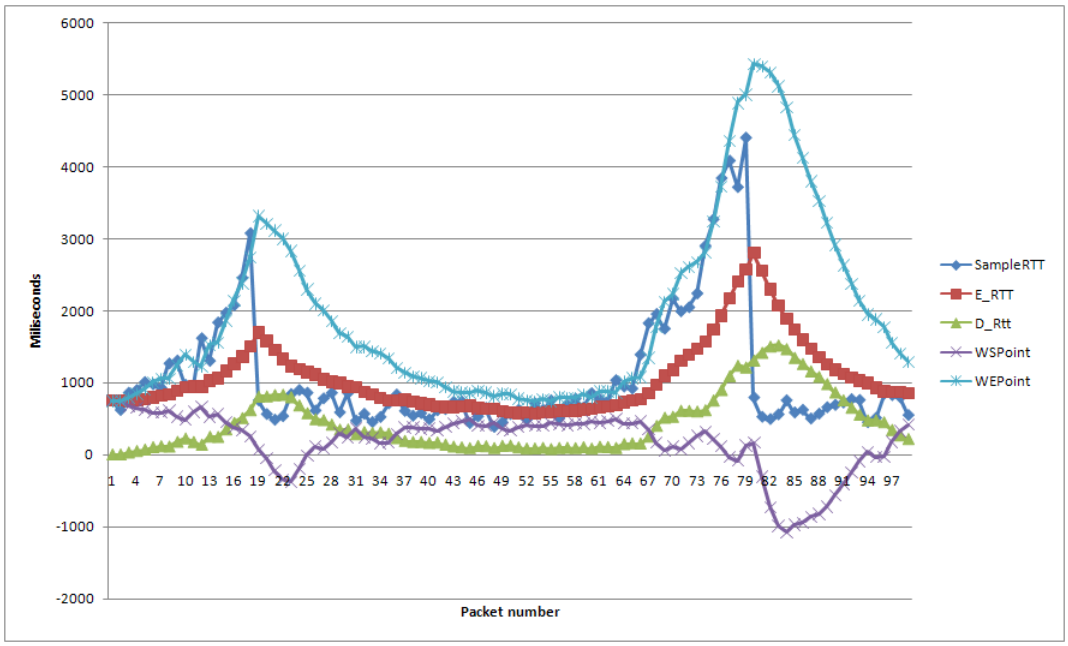

Figure 4.8: Negative starting point

The way to prevent this weakness is to eliminate the 'too slow' packets from the calculation of the deviation. Thus, preventing the temporary lowering of the starting point. However the DFP algorithm must take into consideration the possibility of a rapid change in the network characteristics or topology. Therefore, the 'too slow' packets must be considered when calculating the Estimated RTT. One of the ideas for distinguishing between a single slow packet and a sequence of such packets is to prevent the first few 'too slow' packets influencing the estimations, but after certain number of 'too slow' packets in a row to include them in the calculations, making the new RTTs build up new and correct estimations.

4.5 Memory Consumptions

The main consideration in choosing the configuration for the DFP algorithms is to prevent attacks, for that the starting point should be as tight as possible to the Esti-

mated RTT. However that thought will also create many false alarms (as explained above), those false alarms are packets that the DNS server stores in memory until it is sure that they are valid replies that arrived too soon. Then they are released from the memory and sent to the user (and the cache). In this section we present how different starting point values affect the memory consumption.

In Figure 4.9 we see how many packets are considered 'too fast' in each of the configurations presented earlier in this work. In Figure 4.10 the same data is presented in percentage mode. These figures will help us estimating what is the amount of memory that must be allocated for the DFP algorithm in each of the configurations.

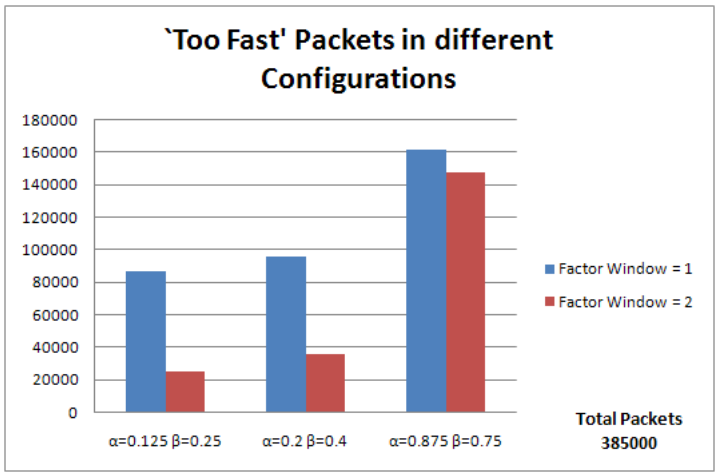

Figure 4.9: Fast Packets in Different Configurations

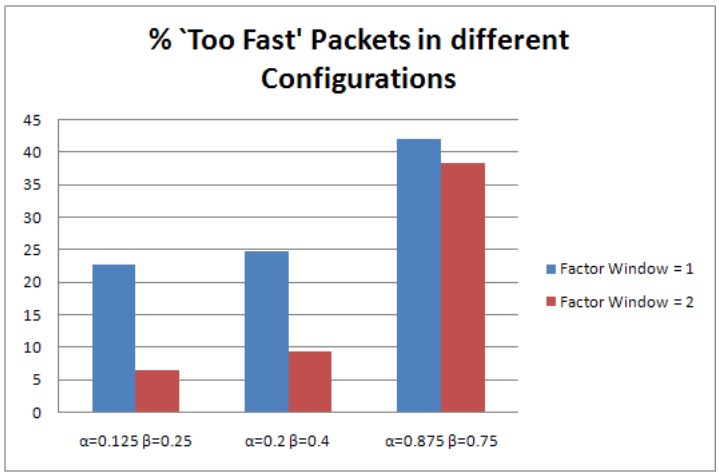

Figure 4.10: Percentage of Fast Packets in different Configuration

Our experiments show that an average reply is about 155 bytes long, the DFP algorithm must allocate those bytes in memory for each delayed packet, usually for about few hundred ms, until the reply is either released or dropped. The memory the DFP algorithm requires depends on how busy is the local DNS server. Figure 4.11 estimates how many KB the DFP algorithm consumes assuming it handles 1000 packets at any given time:

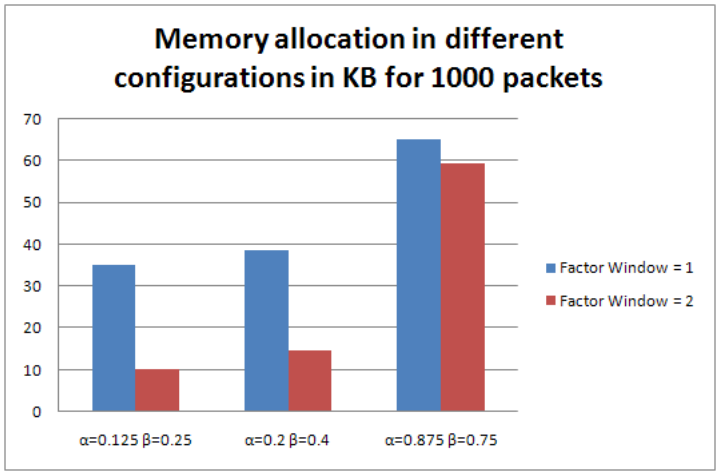

Figure 4.11: Estimation of Memory in KB per 1000 packets

Part II

Simulations

Chapter 5

Experimental Results

5.1 Results of DFP Algorithm

The presented algorithm was implemented and tested on real traffic, collected from HUJI DNS Server. The traffic was sniffed and saved in pcap files, that were later used for different configurations testing and analysis. The traffic was filtered to contain only DNS replies with Authoritative flag on, thus promising that the answers came from an Authoritative serve. See Figure 5.1.

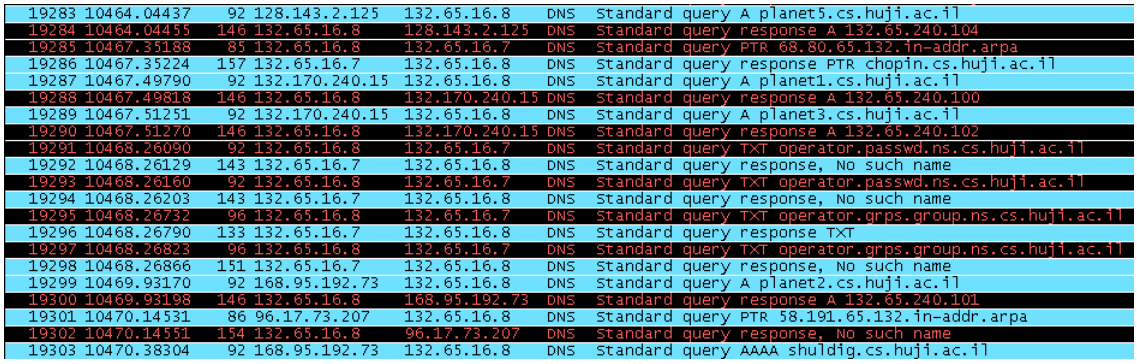

Figure 5.1: Standard pcap file

After the filtering process, the time between the arriving response and the departing request was measured and each of the sampled RTTs was recorded in an appropriate file, according to DNS IP address and DNS query type. See Figure 5.2.

	194.90.221.195	194.90.1.5	192.118.82.168	192.115.106.11
1	3892	4987	6301	9551
2	4159	4608	5595	3877
3	4164	4454	5469	4139
4	4041	5271	5468	4401
5	3900	4652	5447	3791
6	4049	4661	5992	25372
7	4074	5088	7380	5038
8	4026	4824	5459	4092
9	4933	4835	5292	4069
10	3923	4657	10967	4148
11	4010	4387	5238	3177
12	4171	4618	5505	4420
13	3955	4825	8541	3881
14	3973	4322	5692	4004
15	4773	5072	6563	3861
16	4225	4878	5583	4121
17	3930	4898	5779	5143
18	3837	4861	5228	4319
19	4419	4408	5359	4109
20	3938	4495	6136	25411
21	4264	4549	5611	4286
22	4013	4666	5406	22837
23	3861	4735	5456	3684
24	4053	4766	5334	3775
25	3949	4440	5432	4185
26	3992	4818	5919	3890
27	3975	4406	12211	4310
28	3956	4842	6065	4253
29	3937	4265	5418	2936
30	4118	7594	5510	3928
31	3986	5022	5297	3834
32	4139	4440	5896	4097
33	4124	4558	5379	451081
34	3925	4954	5508	3044

Figure 5.2: Sampled RTT (in ms) from the traffic organized by IP addresses and query type

For each of the samples, the algorithm calculates the EstimatedRTT, the De-

vRTT and with given FactorWindow it deduces which packets are considered to be 'too fast'. Short example with 'too fast' packets can be found in Figure 5.3.

	194.90.221.195					192.115.106.11					128.139.6.1			
RTT	E_RTT	D_RTT	FW=1	FW=2	RTT	E_RTT	D_RTT	FW=1	FW=2	RTT	E_RTT	D_RTT	FW=1	FW=2
4159	4125.625	8.34375			3877	4586.25	177.3125	fast	fast	358	374.625	4.15625	fast	fast
4164	4159.203	7.457031			4139	4194.906	146.9609			352	354.8281	3.824219		
4041	4055.775	9.286621	fast		4401	4375.238	116.6611			461	447.7285	6.186035		
3900	3919.472	11.83295	fast		3791	3864.03	105.7533			371	380.5911	7.037292	fast	
4049	4032.809	12.92246			25372	22683.5	751.439			350	353.8239	6.23394		
4074	4068.851	10.97907			5038	7243.688	1115.001	fast		348	348.728	4.857451		
4026	4031.356	9.573397			4092	4485.961	934.7412			345	345.466	3.759588		
4933	4820.295	35.35641			4069	4121.12	714.0859			402	394.9332	4.586379		
3923	4035.162	54.55776	fast	fast	4148	4144.64	536.4044			433	428.2417	4.62937		
4010	4013.145	41.70463			3177	3297.955	432.5421			363	371.1552	5.510829	fast	
4171	4151.268	36.21143			4420	4279.744	359.4705			489	474.2694	7.815772		
3955	3979.534	33.29195			3881	3930.843	282.0636			442	446.0337	6.870248		
3973	3973.817	25.17314			4004	3994.855	213.8339			473	469.6292	5.995383		
4773	4673.102	43.85433			3861	3877.732	164.5584			535	526.8287	6.539375		
4225	4281.013	46.89394	fast		4121	4090.591	131.0209			301	329.2286	11.96168	fast	fast
3930	3973.877	46.1396			5143	5011.449	131.1535			423	411.2786	11.90161		
3837	3854.11	38.8821			4319	4405.556	120.0041			446	441.6598	10.01126		
4419	4348.389	46.8144			4109	4146.07	99.27047			435	435.8325	7.716561		
3938	3989.299	47.93545	fast		25411	22752.88	738.9819			377	384.3541	7.625936		
4013	4040.083	40.1727			4286	6594.36	1131.327	fast	fast	477	465.4193	8.614637		

Figure 5.3: Estimations and fast packets deduction with two FactorWindow options

5.2 Attacks Detection

In order to test the DFP algorithm we have planted few random duplicate reply packets with random arrival time in the tested traffic, the arrival times of the fake replies were shorter than the real replies. The DFP algorithm with the above configuration was able to classify all of the attacks as 'too fast packets' and therefore delayed them until the real result arrived.

We strongly believe that there were no real attempts to attack HUJI local DNS server while the samples were captured, since no duplicate packets were found beside the fake packets planted by us.

Chapter 6

Conclusions

6.1 Future Work

This work focuses on developing a new algorithm for detecting and preventing attacks of the DNS servers. The strength of the algorithm is that it does not rely on any known facts about the attacks themselves (i.e: signature strings, known IP address, etc...). Instead the DFP algorithm relies only on the basic communication and packet identification process in the DNS protocol: the 5-tuple, the Transaction ID and the query type. We strongly believe that this work, implemented as an integrated system on every DNS server, will greatly reduce the probability for a successful attack of the known attacks as well as the unknown zero-day attacks.

Still, further research on possible optimizations for the algorithm is suggested in order to improve its weak points. In the following section we present possible directions for future research.

- **Dynamic configuration**: In the DFP algorithm the three parameters affecting window width are configured only one time, before the algorithm begins. In future research it is worthwhile to find a way to adjust the parameters during the runtime of the algorithm, making the configuration dependant on the characteristics of the network and the expected RTT values. It might be that the dynamic configuration will be able to solve the sudden widening of the window due to the rare and local problems with the RTT of a single packet.

- **Too slow packets**: In the DFP algorithm no special treatment exists to handle the packets that arrive later than the upper bound of the window.

Those packets may be considered as 'too slow' packets and must be treated in a way that would prevent drastic widening of the window. On the one hand, those slow packets can indicate a change in the network so they can not be ignored in the estimation of the RTT. On the other hand, those slow packets are usually just a temporary fault in the Authoritative DNS server or one of the links in the network. Some way must be found that will manage to distinguish between those two cases and treat each one of them accordingly.

Bibliography

[1] www.faqs.org/rfcs/rfc1034.html

[2] www.faqs.org/rfcs/rfc1035.html

[3] http://www.isc.org/software/bind

[4] www.faqs.org/rfcs/rfc791.html

[5] www.faqs.org/rfcs/rfc793.html

[6] www.faqs.org/rfcs/rfc768.html

[7] http://ketil.froyn.name/poison.html

[8] BIND 9 DNS Cache Poisoning, Amit Klein, March-June 2007

[9] http://www.rnp.br/cais/alertas/2002/cais-ALR-19112002a.html

[10] https://www.kb.cert.org/vuls/id/457875

[11] DNS and BIND, 4th Edition, by By Paul Albitz and Cricket Liu, Chapter 11, Security

[12] http://groups.google.com/group/comp.protocols.dns.bind/topics?pli=1

[13] http://tldp.org/LDP/lame/LAME/linux-admin-made-easy/domain-name-server.html

[14] DNS Cache Poisoning: Definition and Prevention, Tom Olzak, March 2006

[15] DNS Spoofing (Malicious Cache Poisoning), Doug Sax

[16] http://www.dnssec.net/

[17] http://slaptijack.com/software/how-to-hide-your-bind-version/

[18] http://linuxgazette.net/153/moen.html

[19] Use of Bit 0x20 in DNS Labels to Improve Transaction Identity, P. Vixie and D. Dagon, March 2008

[20] Solving the DNS Cache Poisoning Problem Without Changing the Protocol, Roberto Perdisci, Manos Antonakakis, and Wenke Lee, May 2008

[21] http://www.topbits.com/securing-dns-servers.html

[22] http://www.ietf.org/rfc/rfc3833.txt