International Journal of Engineering Research (ISSN : 2319-6890)
Volume No.2, Issue No. 5, pp : 357-361 01 Sept. 2013

# An Intrusion Detection System for Kaminsky DNS Cache poisoning

**Dhrubajyoti Pathak, Kaushik Baruah**
Departement of CSE, IIT Guwahati
drbj153@alumni.iitg.ernet.in, b.kaushik@iitg.ernet.in

***Abstract : Domain Name System (DNS) is the largest and most actively distributed, hierarchical and scalable database system which plays an incredibly inevitable role behind the functioning of the Internet as we know it today. A DNS translates human readable and meaningful domain names such as www.iitg.ernet.in into an Internet Protocol (IP) address such as 202.141.80.6. It is used for locating a resource on the World Wide Web. Without a DNS, the Internet services as we know it, would come to a halt. In our thesis, we proposed an Intrusion Detection System(IDS) for Kaminsky cache poisoning attacks. Our system relies on the existing properties of the DNS protocol.***

**Keywords—DNS, DNS Cache poisoning, Kaminsky DNS Cache poisoning, IDS**

## I. Introduction

Cache poisoning in DNS[i][ii] is where an attacker successfully injects bogus data into the recursive nameserver’s cache, causing it to give a reply containing spoofed IP address for a domain to a resolver or client. It is similar to "arp spoofing” where an attacker replies with another MAC address.

The Kaminsky DNS[iii] attack is a recently discovered vulnerability in the DNS protocol[iv] that allows an attacker to effectively poison the DNS cache with little effort, compared to the traditional cache poisoning[v] attacks. Kaminsky attack is a type of brute force cache poisoning attack which allows an attacker to perform continuously with forged DNS response for a domain name without the need to wait for DNS cache timeouts. Unlike the traditional DNS cache poisoning attack the greatest advantage of Kaminsky’s attack is that if the attack does not succeed at the first trial, the attacker can immediately retry, without the need to wait for TTL of the cache to expire. This improves the probability of success of the attack.

DNS security system are generally concerned with cache; DNS cache is inevitable for performance of the DNS system. Kaminsky cache poisoning attack is the main threat to the DNS security to which many currently deployed DNS servers are vulnerable. So in the paper, we concentrate to devise a new technique to cope with the Kaminsky attack.

## II. Material and Methodology

Several schemes such as the DNSCurve[vi], WSEC[vii] DNS, Anax[viii] have been proposed in the literature on improving the DNS security against cache poisoning attacks. However, they require explicit or implicit changes to the DNS protocol and some are limited in providing security.

Some of the world’s top DNS experts decides for a temporary fix against the cache poisoning attack - randomizing the source port[ix]. In this system attacker must correctly guess the unique 16-bit transaction ID along with the 16-bit port number as well. The effective transaction space becomes $2^{16}$ x$2^{16}$ = $2^{32}$ number[x]. Here some of the ports(0-1023) number are fix for some certain application and cannot be used for other purpose[xi].

Sometimes Port randomization used in every outgoing query becomes ineffective by the presence of devices such as routers, gateways or, firewalls or load balancers that perform NAT/PAT which may modify the source ports without preserving the nature of original port randomness[xii].

This section provides the overview of the methodology devised for IDS to detect cache poisoning using an existing DNS infrastructure. When a DNS query response comes to the DNS, the IDS first stores it in the raw DNS data collector. Then it verifies the source IP address and the corresponding hostname in the response packet to be poisonous or legitimate. If the IDS can not verify locally, a resolve request is send to another DNS server (preconfigured, restricted). Details of verified. hostname-IP(RR) mapping are stored in respective tables. Figure-1 shows the flow control of the whole system. Each incoming DNS response packet is given as an input.

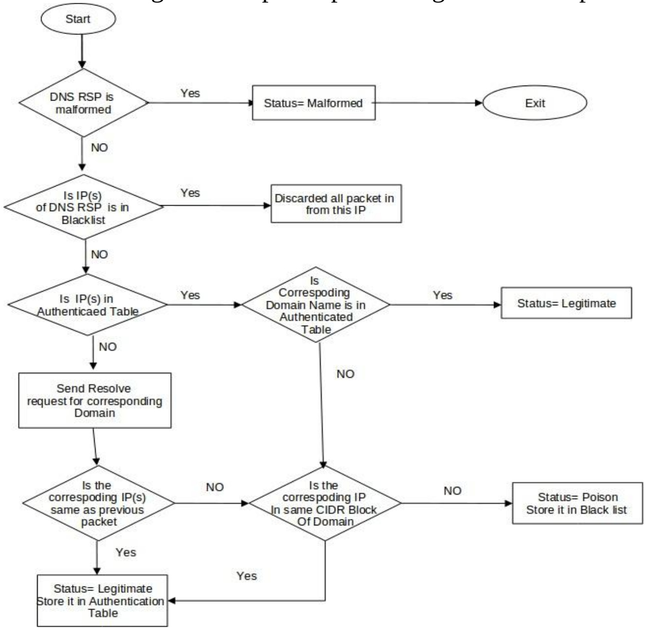
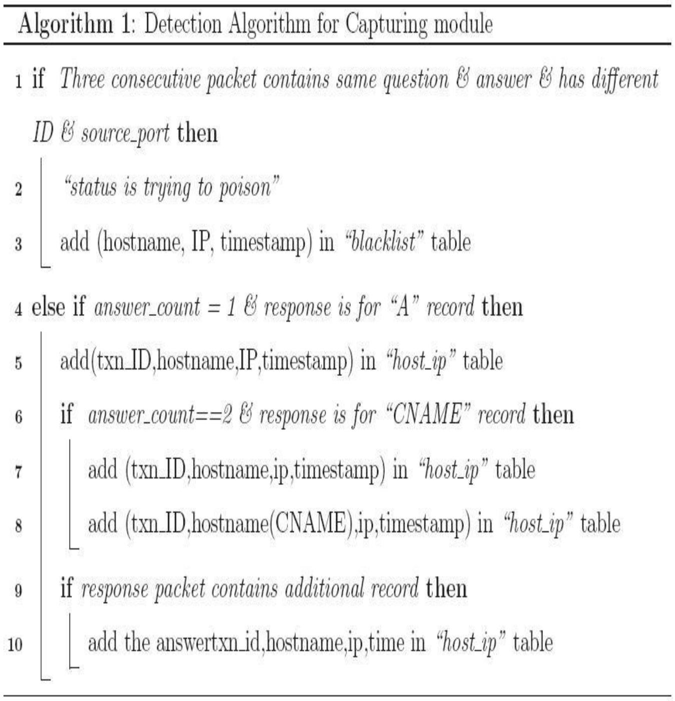

Start
DNS RSP is malformed
Yes
Status= Malformed
Exit
NO
Is IP(s) of DNS RSP is in Blacklist
Yes
Discarded all packet in from this IP
NO
Is IP(s) in Authenticaed Table
Yes
Is Correspoding Domain Name is in Authenticated Table
Yes
Status= Legitimate
NO
NO
Send Resolve request for corresponding Domain
Is the correspoding IP(s) same as previous packet
NO
Is the correspoding IP In same CIDR Block Of Domain
NO
Status= Poison Store it in Black list
Yes
Yes
Status= Legitimate Store it in Authentication Table

Figure 1

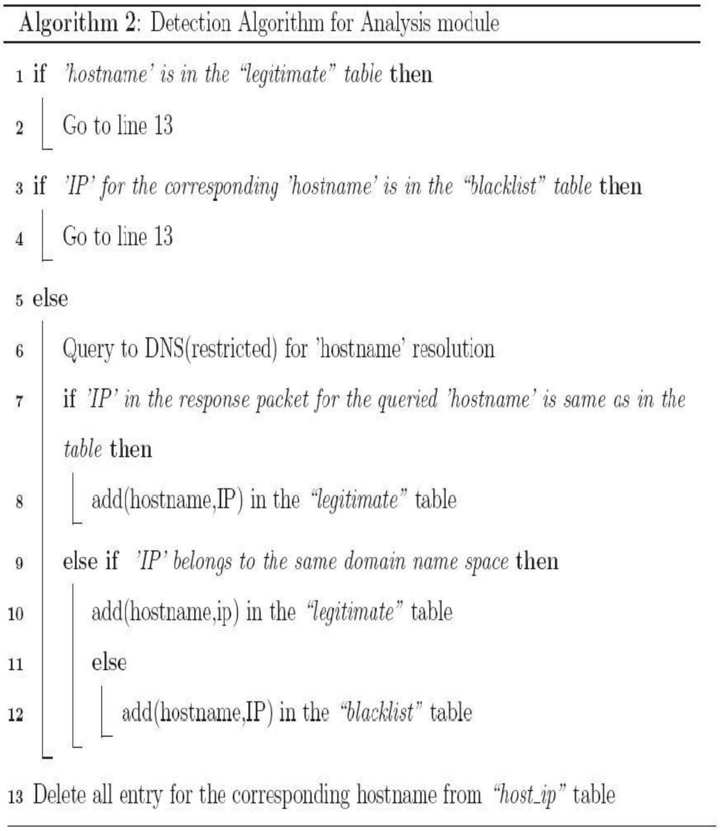
**Detection Model**

The model consists of two part capturing module and Analysis module. The capturing module captures all the DNS response packets received by the DNS server and stores it. The next Analysis module checks the authenticity of the collected records. Algorithm-1 is the used in this work. It waits for a DNS response packet, once it receives a response packet it checks for spoofing (line 1). We label an IP address in the resolving packet for a hostname as spoofed, if at least three consecutive response packets are received bearing the same question and answer pair having different transaction IDs and source ports. Because a DNS will never query for a hostname available in its cache, till the DNS cache has expired (generally 24 hrs.). If the spoofed IP address is labelled as Blacklisted, it is stored in the ”blacklist” table. In line 4 the module checks whether the answer contains an 'A' record, if so then the module stores the transaction ID, hostname and the IP address in the table "host ip". If the response contains a ’CNAME’ record(line 6) then, the module stores both the hostname along with the corresponding CNAME with the same IP address in the ”host ip” table. The module also stores all the records (hostname,ip address) available in the additional section of the response packets, if present.



```
Algorithm 1: Detection Algorithm for Capturing module
1 if Three consecutive packet contains same question & answer & has different
  ID & source_port then
2     “status is trying to poison”
3     add (hostname, IP, timestamp) in “blacklist” table
4 else if answer_count = 1 & response is for “A” record then
5     add(txn_ID,hostname,IP,timestamp) in “host_ip” table
6     if answer_count==2 & response is for “CNAME” record then
7         add (txn_ID,hostname,ip,timestamp) in “host_ip” table
8         add (txn_ID,hostname(CNAME),ip,timestamp) in “host_ip” table
9     if response packet contains additional record then
10        add the answertxn_id,hostname,ip,time in “host_ip” table
```

Algorithm-2 periodically checks the ”host ip” table. If there are records present in the table it retrieves them one at a time. The Line 1 Searches for an entry of the ’hostname’ in the "legitimate" table, if it contains the record. If not, the Line 3 searches for the corresponding ’IP address’ in the ”blacklist” table. The line 6, if the data is not present in both the "legitimate" and ”blacklist” tables then, it send a query to a specified and restricted DNS for a name resolution. Thereafter, it checks the mappings of ’hostname-IP address’ in the response packet received from the specified and restricted DNS with the information available in the ”host ip” table. If there is a match or the IP address resides in same domain name space then, the algorithm labels it as a legitimate one and stores it in the "legitimate" table. Otherwise, the algorithm labels it as blacklist and stores it in the "blacklist" table. After the labelling is done, line 13, delete all entries for the corresponding hostname from the ”host ip” table.



```
Algorithm 2: Detection Algorithm for Analysis module
1 if ’hostname’ is in the “legitimate” table then
2     Go to line 13
3 if ’IP’ for the corresponding ’hostname’ is in the “blacklist” table then
4     Go to line 13
5 else
6     Query to DNS(restricted) for ’hostname’ resolution
7     if ’IP’ in the response packet for the queried ’hostname’ is same as in the
      table then
8         add(hostname,IP) in the “legitimate” table
9     else if ’IP’ belongs to the same domain name space then
10        add(hostname,ip) in the “legitimate” table
11        else
12            add(hostname,IP) in the “blacklist” table
13 Delete all entry for the corresponding hostname from “host_ip” table
```

III. Results and Tables

The Kaminsky class DNS cache poisoning attack effect mostly open recursive DNS servers. Our aim is to detect all those responses that reaches a DNS server containing malicious or poisoned records. By malicious records we mean the “hostname-IP address” mapping is fake. A DNS server has no way to check and verify the “hostname-IP address” mappings. Therefore, the Capture and Analysis module should reside in the recursive DNS server itself. The system runs in parallel with the existing DNS

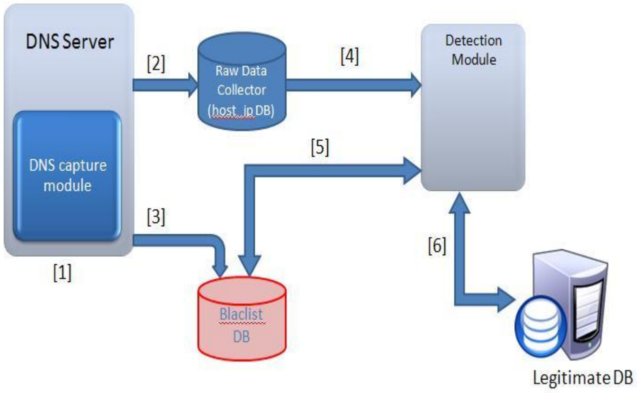


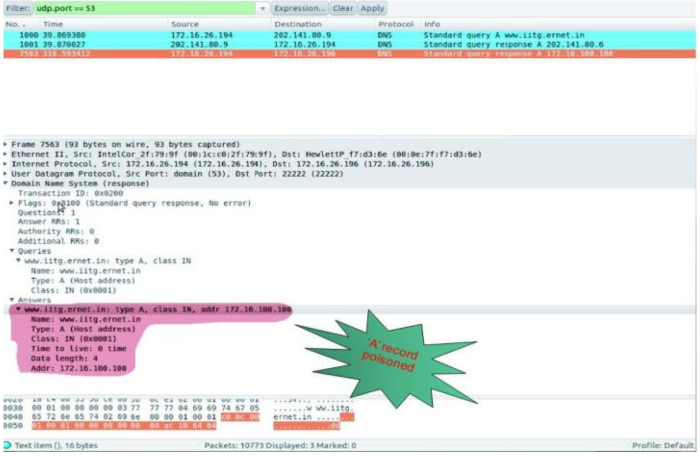
Figure-2

server on the same host. The setup created for the experiment consists of four machines – one as an Authoritative DNS server, one as a Recursive DNS server, one as a DNS client and one as the attacker. We assume that the recursive DNS server does not contain any authoritative record. The basic block diagram for the (software) implementation of the proposed Intrusion Detection System is given in the Figure-2. As shown in the figure, in Step 1 [1], the Capturing module captures each DNS response packet and (Step [2]) stores it in the raw DNS data collector. It also checks for any flooding of DNS responses, i.e., someone trying to poison with a streaming of packets bearing different transaction ID and source port having the same question and answer. If so, then make the RR response Blacklisted in Step [3]. In Step [5] and Step [6], The Analysis module verifies each entry in the Raw data collector, whether it is legitimate or not. If not legitimate, it triggers an RR entry to the "BlacklistDB" else an entry to the "legitimateDB" table respectively.

The Capture Module captures all the DNS response packets off the wire. It collects only the raw DNS packets without any verification. Then it stores it in the Raw Data Collector. The Capture Module does not collect any information about the request packets, as it is not necessary for detection. Since, the DNS implicitly discards all the response packets if and only if it does not contain the same question with the same transaction ID and source port. There are different events that may arise during the detection.

i) When there is no intervention of any third party, i.e., Normal operation. A DNS response packet has shown, where DNS response for the 'A' record in the answer field for the host "www.iitg.ernet.in" is 202.141.80.6, Which is legitimate.

ii) If there is changing of the 'A' or 'CNAME' record in the response field.



```
No. .  Time         Source          Destination     Protocol  Info
  1000 39.869380    172.16.26.194   202.141.80.9    DNS       Standard query A www.iitg.ernet.in
  1001 39.870027    202.141.80.9    172.16.26.194   DNS       Standard query response A 202.141.80.6
  7563 310.593412   172.16.26.194   172.16.26.196   DNS       Standard query response A 172.16.100.100
```

```
▸ Frame 7563 (93 bytes on wire, 93 bytes captured)
▸ Ethernet II, Src: IntelCor_2f:79:9f (00:1c:c0:2f:79:9f), Dst: HewlettP_f7:d3:6e (00:0e:7f:f7:d3:6e)
▸ Internet Protocol, Src: 172.16.26.194 (172.16.26.194), Dst: 172.16.26.196 (172.16.26.196)
▸ User Datagram Protocol, Src Port: domain (53), Dst Port: 22222 (22222)
▾ Domain Name System (response)
    Transaction ID: 0x0200
  ▸ Flags: 0x8100 (Standard query response, No error)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ www.iitg.ernet.in: type A, class IN
        Name: www.iitg.ernet.in
        Type: A (Host address)
        Class: IN (0x0001)
  ▾ Answers
    ▾ www.iitg.ernet.in: type A, class IN, addr 172.16.100.100
        Name: www.iitg.ernet.in
        Type: A (Host address)
        Class: IN (0x0001)
        Time to live: 0 time
        Data length: 4
        Addr: 172.16.100.100
```

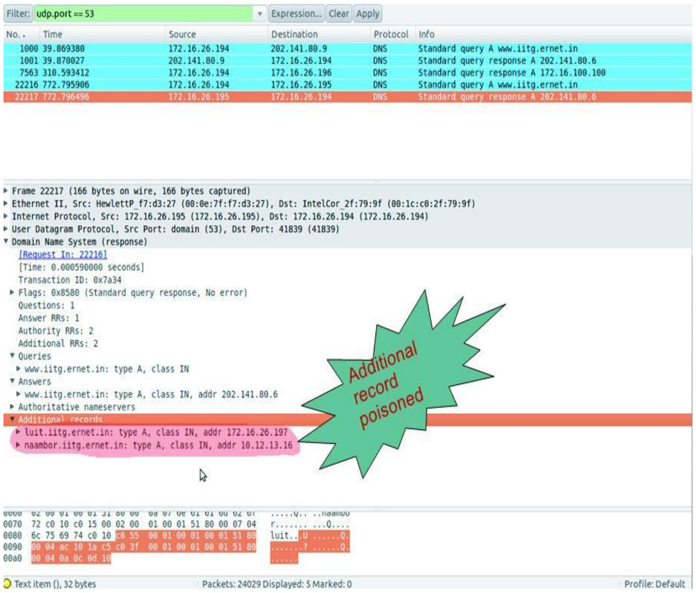
```
0030  00 01 00 00 00 00 03 77  77 77 04 69 69 74 67 05   .......w ww.iitg.
0040  65 72 6e 65 74 02 69 6e  00 00 01 00 01 c0 0c 00   ernet.in .......
0050  01 00 01 00 00 00 00 00  04 ac 10 64 64            ........ ...dd
```

```
Text item (), 16 bytes        Packets: 10773 Displayed: 3 Marked: 0        Profile: Default
```

Figure-3

A spoof/poison DNS response in Wireshark[xiii] has shown in the Figure-3. Here the incoming response packet for 'A' record for domain name "www.iitg.ernet.in" has been sent with fake hostname-IP address mapping in the answer field. If the attacker successfully guess the transaction ID and source port number. Then all the client is redirected to the host '172.16.100.100', which is not the right one. Here the user or DNS has no way to check the mapping.

iii) If there is changing of the mapping of the 'NS' record in the additional field. In the figure-4 a poisoned DNS response has shown, in which the attacker response behave like a normal one, where the answer field contains the legitimate hostname-IP address mapping, but these time he inject fake answer for the 'NS' record. If he success, he can hijack the



```
No. .  Time         Source          Destination     Protocol  Info
  1000 39.869380    172.16.26.194   202.141.80.9    DNS       Standard query A www.iitg.ernet.in
  1001 39.870027    202.141.80.9    172.16.26.194   DNS       Standard query response A 202.141.80.6
  7563 310.593412   172.16.26.194   172.16.26.196   DNS       Standard query response A 172.16.100.100
 22216 772.795906   172.16.26.194   172.16.26.195   DNS       Standard query A www.iitg.ernet.in
 22217 772.796496   172.16.26.195   172.16.26.194   DNS       Standard query response A 202.141.80.6
```

```
▸ Frame 22217 (166 bytes on wire, 166 bytes captured)
▸ Ethernet II, Src: HewlettP_f7:d3:27 (00:0e:7f:f7:d3:27), Dst: IntelCor_2f:79:9f (00:1c:c0:2f:79:9f)
▸ Internet Protocol, Src: 172.16.26.195 (172.16.26.195), Dst: 172.16.26.194 (172.16.26.194)
▸ User Datagram Protocol, Src Port: domain (53), Dst Port: 41839 (41839)
▾ Domain Name System (response)
    [Request In: 22216]
    [Time: 0.000590000 seconds]
    Transaction ID: 0x7a34
  ▸ Flags: 0x8580 (Standard query response, No error)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 2
    Additional RRs: 2
  ▾ Queries
    ▸ www.iitg.ernet.in: type A, class IN
  ▾ Answers
    ▸ www.iitg.ernet.in: type A, class IN, addr 202.141.80.6
  ▸ Authoritative nameservers
  ▾ Additional records
    ▸ luit.iitg.ernet.in: type A, class IN, addr 172.16.26.197
    ▸ naambor.iitg.ernet.in: type A, class IN, addr 10.12.13.16
```

```
0070  72 c0 10 c0 15 00 02 00  01 00 01 51 80 00 07 04   r....... ...Q....
0080  6c 75 69 74 c0 10 c0 55  00 01 00 01 00 01 51 80   luit...U ......Q.
0090  00 04 ac 10 1a c5 c0 3f  00 01 00 01 00 01 51 80   .......? ......Q.
00a0  00 04 0a 0c 0d 10                                  ......
```

```
Text item (), 32 bytes        Packets: 24029 Displayed: 5 Marked: 0        Profile: Default
```

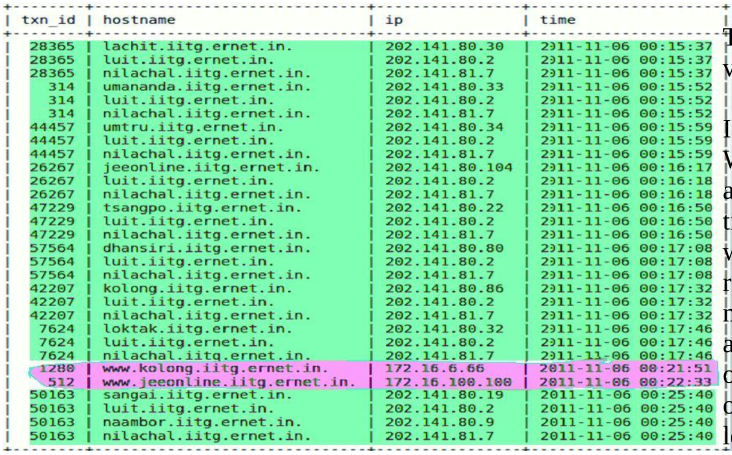
,

Figure-4

whole .iitg.ernet.in domain. Any query for any host in the .iitg.ernet.in domain will goes to his machines (172.16.26.197, 10.12.13.16). Now he can send any mapping for any host in that domain.

iv) When there is continuous response with the same question and answer field with different transaction ID and source port. This means an attacker generating response packet trying to inject his malicious machine address on be half of legitimate.

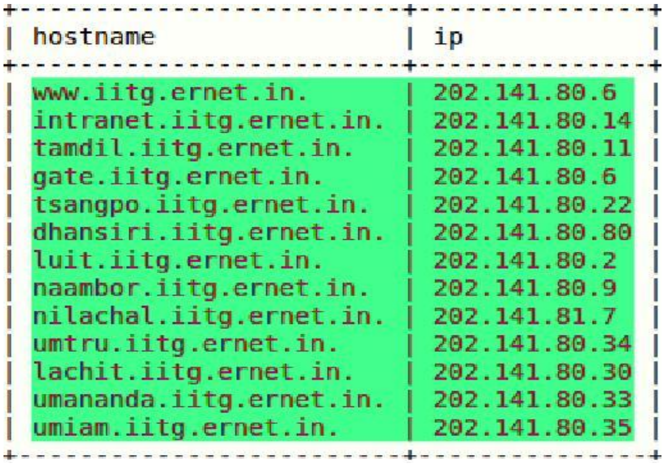
The Figure 5 shows a snapshot of the "host ip" table in the DNS server. The table contains an instance of all the received DNS respones packet by the DNS server. The Capturing module populates the table with the respective field.



```
+---------+-------------------------------+----------------+---------------------+
| txn_id  | hostname                      | ip             | time                |
+---------+-------------------------------+----------------+---------------------+
|   28365 | lachit.iitg.ernet.in.         | 202.141.80.30  | 2011-11-06 00:15:37 |
|   28365 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:15:37 |
|   28365 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:15:37 |
|     314 | umananda.iitg.ernet.in.       | 202.141.80.33  | 2011-11-06 00:15:52 |
|     314 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:15:52 |
|     314 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:15:52 |
|   44457 | umtru.iitg.ernet.in.          | 202.141.80.34  | 2011-11-06 00:15:59 |
|   44457 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:15:59 |
|   44457 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:15:59 |
|   26267 | jeeonline.iitg.ernet.in.      | 202.141.80.104 | 2011-11-06 00:16:17 |
|   26267 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:16:18 |
|   26267 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:16:18 |
|   47229 | tsangpo.iitg.ernet.in.        | 202.141.80.22  | 2011-11-06 00:16:50 |
|   47229 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:16:50 |
|   47229 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:16:50 |
|   57564 | dhansiri.iitg.ernet.in.       | 202.141.80.80  | 2011-11-06 00:17:08 |
|   57564 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:17:08 |
|   57564 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:17:08 |
|   42207 | kolong.iitg.ernet.in.         | 202.141.80.86  | 2011-11-06 00:17:32 |
|   42207 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:17:32 |
|   42207 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:17:32 |
|    7624 | loktak.iitg.ernet.in.         | 202.141.80.32  | 2011-11-06 00:17:46 |
|    7624 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:17:46 |
|    7624 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:17:46 |
|    1280 | www.kolong.iitg.ernet.in.     | 172.16.6.66    | 2011-11-06 00:21:51 |
|     512 | www.jeeonline.iitg.ernet.in.  | 172.16.100.100 | 2011-11-06 00:22:33 |
|   50163 | sangai.iitg.ernet.in.         | 202.141.80.19  | 2011-11-06 00:25:40 |
|   50163 | luit.iitg.ernet.in.           | 202.141.80.2   | 2011-11-06 00:25:40 |
|   50163 | naambor.iitg.ernet.in.        | 202.141.80.9   | 2011-11-06 00:25:40 |
|   50163 | nilachal.iitg.ernet.in.       | 202.141.81.7   | 2011-11-06 00:25:40 |
+---------+-------------------------------+----------------+---------------------+
```

Figure-5

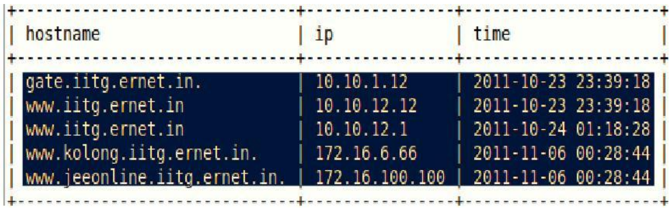
The Figur-6 shows a snapshot of the "legitimate" table in the DNS server. The table contains the record which has been verified and labeled as legitimate. The Analysis module populate this table after verification. In verification phase, the Analysis module also use this table for further verification or detection. If some "hostname-IP" mapping is in this table, then it can directly confirm its status.



```
+-----------------------------+----------------+
| hostname                    | ip             |
+-----------------------------+----------------+
| www.iitg.ernet.in.          | 202.141.80.6   |
| intranet.iitg.ernet.in.     | 202.141.80.14  |
| tamdil.iitg.ernet.in.       | 202.141.80.11  |
| gate.iitg.ernet.in.         | 202.141.80.6   |
| tsangpo.iitg.ernet.in.      | 202.141.80.22  |
| dhansiri.iitg.ernet.in.     | 202.141.80.80  |
| luit.iitg.ernet.in.         | 202.141.80.2   |
| naambor.iitg.ernet.in.      | 202.141.80.9   |
| nilachal.iitg.ernet.in.     | 202.141.81.7   |
| umtru.iitg.ernet.in.        | 202.141.80.34  |
| lachit.iitg.ernet.in.       | 202.141.80.30  |
| umananda.iitg.ernet.in.     | 202.141.80.33  |
| umiam.iitg.ernet.in.        | 202.141.80.35  |
+-----------------------------+----------------+
```

Figure-6

The Figure-7 displays a snapshot of the "blacklist" table in the DNS server. The table contains all the 'hostname-IP' address which mapping is not legitimate or fake. After verification the Analysis module populates this table.



```
+------------------------------+----------------+---------------------+
| hostname                     | ip             | time                |
+------------------------------+----------------+---------------------+
| gate.iitg.ernet.in.          | 10.10.1.12     | 2011-10-23 23:39:18 |
| www.iitg.ernet.in            | 10.10.12.12    | 2011-10-23 23:39:18 |
| www.iitg.ernet.in            | 10.10.12.1     | 2011-10-24 01:18:28 |
| www.kolong.iitg.ernet.in.    | 172.16.6.66    | 2011-11-06 00:28:44 |
| www.jeeonline.iitg.ernet.in. | 172.16.100.100 | 2011-11-06 00:28:44 |
+------------------------------+----------------+---------------------+
```

Figure-7

Thereafter the analysis module use this table for further verification of DNS response record.

## IV. Conclusion

We have witnessed that the Kaminsky DNS cache poisoning is a threat to the Recursive Domain Name System, at the same time there are no security measures currently in place to cope with this type of attack. Also, the DNSSEC deployment requires a good understanding of managing cryptography, key management and coordination across various DNS administrative domains. These are non-trivial issues to overcome and will take time before DNSSEC is fully deployed on the Internet. Therefore, we have proposed an independent, locally deployable IDS for detecting traditional DNS cache poisoning attacks along with the Kaminsky DNS cache poisoning attack. Our approach does not require any changes to the existing Domain Name System and relies on the fundamental infrastructure of the Domain Name System. Moreover, it works in a fully automated manner. The proposed IDS was deployed in a LAN environment and the result showed successful detection of DNS cache poisoning attacks. Also, the experimental results illustrated the fact that, extra DNS request packets are generated due to the step involving verification of the DNS response records in our implementation. In this scheme, we only label the records and store it in the respective databases. We assume remedial action against the attacker will be handled by the DNS administrator.

## Acknowledgement

We take the opportunity to render my deep sense of gratitude and tribute to Dr. Santosh Biswas, Assistant Professor, Dept. of CSE, IIT Guwahati for his constant and valuable guidance in the truest sense throughout the course of work.

## References

i. P. Mokapetris. Domain Names concept and facilities,november,1987.http://www.ietf.org/rfc/rfc1034.txt.

ii. P. Mokapetris. Domain names implementation and specification, November 1987.http://www.ietf.org/rfc/rfc1035

iii. DNS Cache Poisoning Vulnerability. www.iana.org/about/.../davies-viareggioentropyvuln-081002.pdf

iv. J. Stewart. DNS Cache poisoning thenextgengeration. http://w.secureworks.com/research/articles/dns-cache-poisoning/, 2002

v. D. Kaminsky. Black ops 2008- its the end of the cache as we know it. Presented at BlackHat 2008, http://www.slideshare.net/dakami/dmk-bo2-k8, 2008

vi. Daniel J. Bernstein, http://dnscurve.org

vii. R. Perdisci, M. Antonikakis, X. Luo, W. Lee. WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning attacks. In proceeding of DSN-DCCS, Estoril, LIspon, July 2009

viii. M. Antonakakis, D. Dagon, X. Luo, R. Perdisci, W Lee, J. Bellmor. A cetralized Monitoring Infrastructure for improving DNS Security. RAID 2010, LNCS 6307, pp. 18-37, 2010.

ix. Measures for making DNS more resilent against forged answers, July 2008. http://tools.ietf.org/html/draft-ietf-dnsext-forgery-resilience-06.

x. S. Friedl. An Illustrated guide to Kaminsky DNS vulnerability,http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

xi. Port Numbers, http://www.iana.org/assignments/port-numbers

xii. C. R. Dougherty. Vulnerability Note VU#800113,http://www.kb.cert.org/vuls/id/800113

xiii. Wireshark-Go Deep, http://www.wireshark.org/