67607 -Kaminsky's Attack

Instructed by Amit Klein, written by Agam Ebel
(Based on a work by Ilay Bahat and David Keisar Schmidt)

2025

Contents

1 Kaminsky's DNS Cache Poisoning Attack

1.1 Introduction

DNS cache poisoning aims to make a DNS resolver return an incorrect response so that users who resolve names to addresses through this DNS server and then connect to the IP addresses returned from it are directed to an attacker IP address. We do that by inserting false information into the DNS resolver cache.

One of the most significant DNS Cache poisoning attacks is Kaminsky's Attack, which was presented at Black Hat USA 2008 in the lecture "Black Ops 2008: It's The End Of The Cache As We Know It" – `https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf`.

In this exercise, your goal is to poison BIND9's cache so that it will resolve (from cache) the name `www.example1.cybercourse.example.com` into the IP address `6.6.6.6`, using a Kaminsky-style attack.

1.2 Our Environment

Our BIND9 DNS server is vulnerable to cache poisoning and deployed within a Docker container.

We also have an authoritative name server deployed within a Docker container. It will be used as a "root" name server. We limit our work only to domains that are subdomains of `cybercourse.example.com`. This domain serves as our effective root, and **no record outside this domain should ever be attacked in this exercise**. Moreover, all containers are connected to an isolated network, so trying to request any record outside this domain (`cybercourse.example.com`) will fail.

The attacker will control two components: an authoritative name server and an attacker client (each component is deployed as a separate docker container). The client will try to poison BIND9's cache by sending spoofed DNS responses.

We also deployed a victim client within a docker container. It will be used to check if our attack was successful by sending a request to `www.example1.cybercourse.example.com` and see if we got our poisoned record as an answer (for example, we can do it by issuing a `ping` request to the domain and observe to which IP the request was sent).

All the containers are connected to a virtual LAN named `k-net`, this is an isolated network where we operate.

1.3 General Information

- To log in to the environment, use username "KEnv" and password 5260
- All of the components are deployed within docker containers and have the following IP and MAC addresses:

Container name	IP address	MAC address
vulnerable BIND9 DNS resolver	`192.168.1.203`	`02:42:ac:11:00:03`
"effective root" authoritative name server	`192.168.1.204`	`02:42:ac:11:00:04`
attacker's authoritative name server	`192.168.1.201`	`02:42:ac:11:00:01`
attacker's "client"	`192.168.1.202`	`02:42:ac:11:00:02`
victim client	`192.168.1.205`	`02:42:ac:11:00:05`

- The containers are connected to an isolated virtual LAN named `k-net`. **Note that the containers are connected only to this network, so when checking and developing the attack, do it from inside the containers only!**
- The attack target is a BIND9.4.1 DNS caching server.
- It is assumed that the domain (name) we try to poison (i.e. we try to inject a record for) is not yet cached (or that its cache entry has expired).
- Please note that we also added a time delay for every outgoing packet from the root name server on purpose, to make sure we have a sufficiently long "attack window" before the genuine answer arrives, during which we can inject our spoofed packets.
- We can use LDNS library to simplify creating and processing DNS packets.

Note – You can work on the source code file from within the VM (you should place the attacker's authoritative name server files under the directory `/home/kenv/Desktop/attackerAuth` on the VM), and you'll be able to access it in the attacker's authoritative name server container under `/tmp/attackerAuthNS` (which is mounted to the `/home/kenv/Desktop/attackerAuth`). For the attacker's client files, you should place the files under the directory `/home/kenv/Desktop/Mclient` on the VM), and you'll be able to access it in the attacker's client container under `/tmp/mclient` (which is mounted to the `/home/kenv/Desktop/Mclient`).

Therefore, it is enough to change the source code itself inside the `attackerAuth` directory on the VM (or in the case of the attacker's client source code, it is enough to change the source code itself inside the `Mclient` directory on the VM) and the changes will also apply in the right container.

You must compile the source code from inside the right container (with the following command: `gcc -Wall -Wextra -Werror -Wconversion` *`attacker-file.c`* `-lpcap -lldns -o` *`compiled-file-name`*) and ensure that there are no compilation warnings, and run the executable from the containers. This is how we will compile and run your code ourselves (this guarantees you're working with the same compilation and library environment we use), and we will take off points for compilation warnings (and lots of points if it fails to compile or do the job).

1.3.1 Kaminsky's Cache Poisoning Attack Reminders

This attack is the one of the most significant DNS Cache poisoning attacks. It changed the approach on how to perform DNS cache poisoning. With the Kaminsky's attack, when one attempt failed, instead of waiting the TTL for the next attack attempt, we can immediately try to poison again using another subdomain of the original domain we want to poison.

To do so, we will send our spoofed responses in "attack windows". An attack window is defined as the time from when the DNS query is sent from the recursive resolver to the authoritative name server and until the recursive resolver gets the genuine answer (basically it's the RTT between the recursive resolver to the authoritative name server).

According to the Kaminsky's attack, in each attack window we need to use another subdomain name and sync the spoofed responses accordingly.

For example, suppose we want to poison `www.example.com`. Then instead of forcing the resolver to directly resolve this name and sending the spoofed packets for this name/query, we can force the resolver to query an arbitrary (non-existing) subdomain name of `example.com` (e.g. `ww1.example.com`) and in the spoofed answer, add some additional records to carry out the poisoning in an "indirect" manner.

1.3.2 Notes about DNS records

Recall that we have many types of DNS resource records (from now on, we will refer to them as "RR"). Two useful types that we will use in this attack are `NS` records and `A` records.

The **wire**[1] structure of a DNS RR is the following: `<name> <class> <type> <ttl> <rdlength> <radata>`. For example, an `A` record for `www.example.com` looks like this: `www.example.com IN A 500 4 10.20.30.40` (notice that this is not the real IPv4 address of `www.example.com`, but a made-up one only for demonstrating; also notice that we write the wire data in human readable format, on the wire it'll be serialized as `\x03 www \x07 example \x03 com \x00 \x00 \x01 \x00 \x01 \x00 \x00 \x01 \xF4 \x00 \x04 \x0A \x14 \x1E \x28`).

`A` records are a fundamental component of the Domain Name System (DNS). Those records provide the IPv4 address of a given domain (for IPv6 addresses, we similarly use `AAAA` records), and the most common usage of those records is IP address lookup (matching

[1] i.e. as it appears on the network

a domain name to an IPv4 address), there are also less common usages for this kind of record which we will not mention here.

`NS` records are another fundamental component of the Domain Name System (DNS). Those records designate the name server (by its name) which is responsible (authoritative) for the domain provided in the record's key. Without properly configured NS records, users will be unable to load a website or application.
For example, an `NS` record for the domain `example.com` looks like this: `example.com. IN NS 500 ns1.example.com.`

To read more about DNS records, please see `https://www.ionos.com/digitalguide/hosting/technical-matters/dns-records/`

1.3.3 Notes about DNS flags

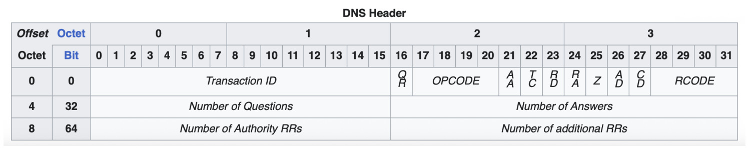
DNS flags are a 16 bit value in the DNS header, which is divided into 8 fields: `QR, opcode, AA, TC, RD, RA, Zero` and `Rcode`. In the following section we provide a short explanation on some of the fields.

`QR` – A 1-bit subfield which determines the message type, if set to 0 the message is of request type and if 1 then the message is of answer type.

`AA` – "Authoritative Answer". A 1-bit subfield which specifies if the server is authoritative for the requested record. The value is 1 if authoritative, else (including in queries), the value is 0.

`RD` – "Recursion Desired". A 1-bit subfield which specifies if the server needs to answer the query recursively. When a server answers, it copies this field from the query.

`RA` – "Recursion Available". A 1-bit subfield which specifies the availability of recursive response. Set to 0 in queries.



DNS Header

Offset	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Transaction ID																QR	OPCODE				AA	TC	RD	RA	Z	AD	CD	RCODE			
4	32	Number of Questions																Number of Answers															
8	64	Number of Authority RRs																Number of additional RRs															

Figure 1: DNS header (picture taken from Wikipedia)

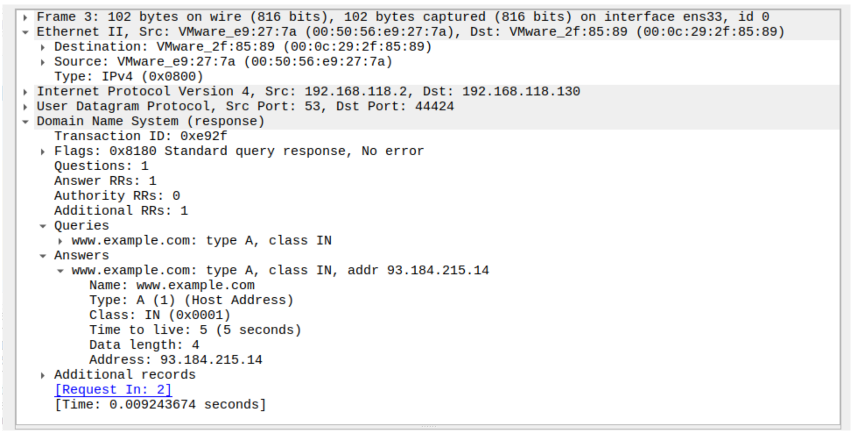
To read more about DNS flags, please see `https://www.geeksforgeeks.org/dns-message-format/`

1.3.4 Notes about Wireshark

As stated on the Wireshark website: “Wireshark is a network packet analyzer. A network packet analyzer presents captured packet data in as much detail as possible.”
Wireshark is a free, open-source network analyzer. One can download it for Windows, Linux, and macOS from here (note that we already installed Wireshark on the exercise environment).

We will use it to analyze the traffic in our virtual LAN (mostly focusing on DNS packets), build and debug the attack attempts. Herein we provide a short introduction to using Wireshark.

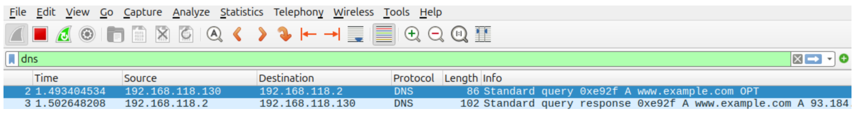
When launching Wireshark, we start by choosing an interface to capture packets from. After we choose an interface, we can see which packets are sent on this network and the content of each packet. For example, here is a screenshot of packet details for a captured response to a DNS query for `www.example.com` we sent earlier (we can see the details of a packet by double-clicking on the packet line – the packet details will appear in a pop-up window) as can be seen here:



```
▸ Frame 3: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface ens33, id 0
▾ Ethernet II, Src: VMware_e9:27:7a (00:50:56:e9:27:7a), Dst: VMware_2f:85:89 (00:0c:29:2f:85:89)
  ▸ Destination: VMware_2f:85:89 (00:0c:29:2f:85:89)
  ▸ Source: VMware_e9:27:7a (00:50:56:e9:27:7a)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.118.2, Dst: 192.168.118.130
▸ User Datagram Protocol, Src Port: 53, Dst Port: 44424
▾ Domain Name System (response)
    Transaction ID: 0xe92f
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 1
  ▾ Queries
    ▸ www.example.com: type A, class IN
  ▾ Answers
    ▾ www.example.com: type A, class IN, addr 93.184.215.14
        Name: www.example.com
        Type: A (1) (Host Address)
        Class: IN (0x0001)
        Time to live: 5 (5 seconds)
        Data length: 4
        Address: 93.184.215.14
  ▸ Additional records
    [Request In: 2]
    [Time: 0.009243674 seconds]
```

Figure 2: Packet content in Wireshark

We can also apply a “display filter” to show only certain types of packets (such as DNS, UDP, TCP...) and packets that were sent to/from specific ports. For example, for the DNS request we sent in the previous example, we can add a filter only for DNS packets, as we can see here:


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
dns

```
  Time         Source           Destination      Protocol Length Info
2 1.493404534  192.168.118.130  192.168.118.2    DNS          86 Standard query 0xe92f A www.example.com OPT
3 1.502648208  192.168.118.2    192.168.118.130  DNS         102 Standard query response 0xe92f A www.example.com A 93.184
```

Figure 3: Packet capturing with filter

For further reading about Wireshark display filters, please refer to `https://www.wireshark.org/docs/man-pages/wireshark-filter.html`

1.4 Starting the environment

Our attack environment consists of a VM and 5 containers. First, download the environment at the following link.
Note – the environment disk requires at least 35GB free on your computer, make sure to have enough free space to upload the disk.

Start the virtual machine using the instructions for uploading a virtual disk to a VM which we provided in the "Environment Setup" file.
Next, run `./run_root_container.sh` which is located in `/home/kenv/Desktop/` to start our "root name server" container and start the "named" daemon of the name server (to enable the resolution process) using the following instructions:

1. `cd Desktop`
2. `./run_root_container.sh`
3. `named -4` (to start the "named" daemon. Notice that we run it from inside the container)
4. `lsof -i4UDP:53 -n -P` (to make sure our server is listening. Notice that we run it from inside the container). The output should be 2 lines which indicated the open network sockets that the server listening on, one is the loopback interface and the other one is our "k-net" docker network.

To start the BIND9.4.1 recursive resolver:

1. `docker start bind9res`
2. `docker exec -it bind9res /bin/bash`
3. `named -4` (to start the "named" daemon. Notice that we run it from inside the container)
4. `lsof -i4UDP:53 -n -P` (to make sure our server is listening. Notice that we run it from inside the container). The output should be 2 lines which indicated the open network sockets that the server listening on, one is the loopback interface and the other one is our "k-net" docker network.

To start the Attacker's authoritative name server:

1. `docker start attacker-auth`
2. `docker exec -it attacker-auth /bin/bash`

To start the Attacker's client:

1. `docker start attacker-client`

2. `docker exec -it attacker-client /bin/bash`

To start the "legitimate client": (which we will use only for checking our attack)

1. `docker start client`

2. `docker exec -it client /bin/bash`

1.5 Kaminsky's Attack – attack reconstruction guidelines

This attack contains many levels and could be hard to follow. For your convenience, we added the attack flow you should follow in your implementation.

Reminder – the server we are attacking is a Bind9.4.1 recursive resolver (which denoted as "bind9res" on our environment) and the domain we are attacking is `www.example1.cybercourse.example.com`.

1. Send a request for `www.attacker.cybercourse.example.com` from the attacker's client to the recursive resolver, its required to get the port which Bind9.4.1 (the recursive resolver) uses to send packets with.
   Note: `attacker.cybercourse.example.com` is the attacker's domain, the root will delegate to the attacker's server IP. You need to implement the server logic there (and submit it also).

2. For each round of the attack, send a query for a subdomain (e.g. `ww1.example1.cybercourse.example.com`) under the domain name we are try to poison.

3. During the time the query is "in flight", the attacker's client tries to spoof the answer by sending the round's spoofed DNS answers (containing the "Kaminsky-style" payload) to the BIND9 recursive resolver.

4. To check if the attack was successful, the client then asks for `www.example1.cybercourse.example.com` and sees if the returned IP by the resolver is `6.6.6.6`. We can use `dig` for sending DNS requests to the recursive resolver for the name `www.example1.cybercourse.example.com`. Use `dig @192.168.1.203 www.example1.cybercourse.example.com` (Use this to review your attack after all rounds have been completed).

Note – please consult the lecture notes for the attack payload.

1.6 Tips and Tricks

- When debugging, do not forget to flush the recursive resolver cache between one attack attempt to another (but **not** between rounds of the same attack! that would be "unfair" because the attacker has no such flushing capability in real life!). We can do that using `rndc flush` inside the recursive resolver container.

- You can use LDNS library for creating DNS requests and responses packets (notice that it is already installed in the attacker's authoritative name server and client containers).
- There are many ways to send packets to the different nodes in the network, some of them are the use of `raw sockets` or `libpcap` (for injecting packets inside the network. Notice, that you will need to build the packet from layer 2 if you choose `libpcap`).
- Use Wireshark to monitor and inspect the packets you send over the virtual LAN. It can help you debug and build your attack attempt.
- To listen to the traffic on our virtual LAN with Wireshark from the VM, we first find the ID of our docker network using `docker network inspect k-net`, check what the ID value, and choose the `br-`*`k-net network ID`* interface to listen on.
- From within the containers, the interface we will send our traffic on is `eth0`.
- When developing the attack attempt, we recommend the students building it incrementally. Start by sending a "real" packet (not part of the attack flow) and ensure it works as intended. Next, implement the attacker's name server and the attacker's client logic. Finally, integrate all the components to create the complete attack flow.
- Test your code! Try to run your attack on the provided server by sending the spoofed packets and checking the output on the server container (using `dig`) or in the client itself when sending another request to the resolver for the domain `www.example1.cybercourse.example.com`. Ensure this is consistent by running several (dozens) attack cycles (between each attack cycle, do not forget to clean the BIND9.4.1 resolver cache by using `rndc flush`.)
- To check your attack before submitting, you can check that you get your spoofed IP when sending a request to `www.example1.cybercourse.example.com` from the "legitimate client" container (i.e. we can do it by sending a request to the domain using `ping` and make sure we get the spoofed address (`6.6.6.6`) instead of the legitimate address). Using `ping` ensures that the name is resolved in the client using the client stub resolver (which is different than `dig`). This is the ultimate check, which is similar to what we do in our acceptance tests. Using `dig` is easier for debugging because dig doesn't have a local cache.
- We can see the genuine IP address that `www.example1.cybercourse.example.com` is mapped to by sending a DNS request using `dig` to the "root" name server.
- You may **not** assume that the `dig` command exists in the attacker's authoritative name server and client containers.
- The spoofed packets should win the race against the genuine authoritative name server but also should arrive at the resolver **after** the query for the domain we try to poison is sent.

- For our automation tests, **we require that you use socket options** (on the TCP socket you use for the control channel) using `setsockopt` and add the following flags: `SO_REUSEADDR`, `SO_REUSEPORT`.
  For your convenience, you can use the following code example:

  ```
  int opt = 1;
  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt,
          sizeof(opt)) < 0) {
      printf("setsockopt(SO_REUSEADDR) failed...\n");
      close(sockfd);
      exit(1);
  }

  opt = 1;
  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, &opt,
          sizeof(opt)) < 0) {
      printf("setsockopt(SO_REUSEPORT) failed...\n");
      close(sockfd);
      exit(1);
  }
  ```

  Note – this requirement is to ensure that our tests will work properly on your submission.

1.6.1 Tips for Debugging

1. For debugging purposes, we can use the `netstat` command for finding open UDP ports in our resolver.

2. For debugging purposes, we can use `dig` for sending DNS requests to some resolver/name server in the network. Use `dig @`*`IP-addr-of-DNS-server the-desired-site`*.

1.7 Submission

1.7.1 Important Remarks

Please follow these instructions carefully to ensure full credit for your submission. Failure to meet any of these requirements will result in a deduction of points.

- When using a TCP socket to send your queries, use socket optimizations using `setsockopt` and add the following flags: `SO_REUSEADDR`, `SO_REUSEPORT`.

- You are required to compute the UDP checksum accurately, and specifically, **not** to set it to zero.
- Make sure to send at most 65536*20 spoofed packets in your attack attempt! We will take points off for violations.
- When compiling your attack files, use `gcc -Wall -Wextra -Werror -Wconversion` *`program-name`*`.c -lldns -lpcap -o` *`desired-compiled-file-name`*
  Compile your source code from within the right container (the attacker's authoritative name server code from the attacker's authoritative name server container and the attacker's client code from the attacker's client container), and make sure there are no compilation warnings! We will take points off for any compilation warnings.
- When running the compiled code, run it without any command line arguments (for each of your files).
- Make sure your code does not print anything to the screen before submitting your attack attempt.
- The exit code from the attacker's client and attacker's authoritative name server code should be 0.
- Running `shell` code form inside the C file (e.g. using `system()`/`execve()` etc.) is strictly prohibited.
- After you submit the exercise in Moodle, download your submission and make sure the files in the submission are correct.
- Please clean up your code before submission – remove redundant code, add useful comments, use reasonable names for variables and functions, etc. – but please double-check that everything still works afterwards!
- Please pay attention to file names (typos...) and do not submit any folders or extra files – just a "flat" zip file with following five files only.

1.7.2 Submission

The submission is done via Moodle in pairs. Please submit a **single zip file named "ex2.zip"** which contains the following files:

1. Your attacker's client code (in `C`). The file should be named "ex2_client.c".
2. Your attacker's authoritative name server code (in `C`). The file should be named "ex2_server.c".

3. A file containing a short explanation (1-2 paragraphs) of their attack attempt, how they built the attacker's authoritative name server, and the attacker's client.
   The file should be named "explanation.txt".

4. A TXT file contains a single line with the submitters' IDs (separated by a comma. i.e. 123456789,987654321). If you submit the exercise alone (after getting approval from the course staff), enter your ID only (without a comma, i.e. 123456789).
   The file should be named "readme.txt".

5. A TXT file with a report if used LLMs for building the code. If so, how and what kind of prompts you gave the LLMs.
   The file should be named "llm_report.txt".

1.8 Resources

- `https://www.cloudflare.com/learning/dns/dns-records/dns-a-record/`
- `https://www.cloudflare.com/en-gb/learning/dns/dns-records/dns-ns-record/`
- `https://www.ionos.com/digitalguide/hosting/technical-matters/dns-records/`
- `https://www.nlnetlabs.nl/documentation/ldns/design.html`
- `https://www.nlnetlabs.nl/projects/ldns/about/`
- `https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html`
- `https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-12`