2IC80 - Kaminsky Vulnerability

Bob de Brabander (0937235), Filip Davidovic (1034603)

April 11, 2018

Attack description

The attack we are performing is called DNS-cache poisoning. There exists multiple versions of this attack, but we are specifically carrying out an attack that is using the Kaminsky Vulnerability found by Dan Kaminsky in the summer of 2008. What we do in this attack is fake a response of an authoritative DNS-server to make it cache an NS-record (authoritative domain) of our choice linked to an A-record of our choice (IP-address). The fake response has to meet the following requirements:

1. It should have the same question as the original query of the recursive DNS to the authoritative DNS
2. It should have the destination port set to the source port of the original query of the recursive DNS to the authoritative DNS
3. It should have the same query-id as the original query of the recursive DNS to the authoritative DNS
4. It should be an authoritative response
5. It should arrive before the actual response

Meeting requirements 1, 4, and 5 is not so hard as we know the question (we sent it), making an answer authoritative only requires you to flip a bit, and we answer instantly without any calculations so our fake response will most likely arrive before the actual response. Meeting requirements 2 and 3 is harder as the query-id has 2^{16} possible values and the port has 2^{11} possible values, leaving 2^{27} possible combinations. We implemented 2 different methods to deal with these problems. The first implementation, the 'loud' implementation, just tries to guess the query-id (in this setup we turned off port randomization leaving only problem 3). The second implementation, the 'silent' implementation, ARP-spoofs the recursive DNS server and the authoritative DNS server (assuming they are in the same network) and simply reads the traffic in between these servers making it trivial to answer a correctly formed response.

Given that:

- the victim DNS's IP-address is a
- our IP-address is b
- the zone we are trying to spoof is z
- the IP-address we want to put in the victim's cache for zone z is c

The loud attack is divided in the following steps:

1. find out the IP-addresses of the servers that are authoritative for zone c and put it in a list l
2. send a DNS-query to a asking where is 'somerandomstring'.z
3. we know now that a is going to ask some server(s) of l where 'somerandomstring'.z is with some query-id q, so we answer (with the same question, a random query-id and the authoritative bit set to 1) that we do not know where 'somerandomstring'.z is but you can ask any of the following servers: all servers in l with in the glue length(l) times the ip of c
4. Now we ask a for the NS-record of any of the servers in l if it responds c we were successful in guessing the query-id, if it answers something else we return to step 2

SIDENOTE: We repeat step 3 multiple times (about 10 times) because fake responses are ignored and we have time to spoof the cache until the actual response arrives (which is usually about 10 packets talking from experience).

The silent attack is divided in the following steps:

1. find out the IP-addresses of the servers that are authoritative for zone c and put it in a list l
2. we ARP-spoof all addresses in l to a with b's MAC-address
3. send a DNS-query to a asking where is 'somerandomstring'.z
4. we know now that a is going to ask some server(s) of l where 'somerandomstring'.z is with some query-id q, but it will actually ask us (ARP-spoofed)
5. we read the packet and send back an authoritative answer pretending to be the same server a contacted with query-id q saying we do not know where 'somerandomstring'.z is but ask c who is authoritative for z
6. a now thinks c is authoritative for z and it will put it in cache as we sent an authoritative answer.
7. if this was successful, a will now ask c where is 'somerandomstring'.z

Usage

The main script file is named `kamins.py`. There are several custom modules that the main script utilizes in order to work. They are in no particular order:

- `vars.py` - module with auxiliary variables. It contains the variables used to display the terminal text in appropriate colors. This module might be populated with more variables in the future versions of the script.
- `validinp.py` - module used to validate the input given in the arguments upon calling the script.
- `utils.py` - module with auxiliary methods used by the script. There are methods to generate a random port, generate a random subdomain, generate a random TXID, generate a random IPv4 address & get the MAC address given an IP.
- `kaminskySilent.py` - module for the execution of the silent attack variant. This module is explained in detail in section 4.3.
- `kaminskyLoud.py` - module for the execution of the loud attack variant. This module is explained in detail in section 4.2.
- `getsoa.py` - module used to fetch the domains and their respective IPs for the authoritative nameservers for the target domain supplied in the arguments upon calling the script.
- `arppoison.py` - module used to ARP cache poison two victims, providing Man-In-The-Middle position for the attacker. It is worth noting that this module assumes that all of the hosts are in a local network.

The following additional Python modules must be installed on the machine that is executing the script in order for it to function properly:

- `scapy`
- `argparse` - used to provide argument input on script execution.
- `validators` - used to validate the IPv4 addresses and domains provided in the argument input.

To execute the attack navigate to the directory where the scripts are located and execute the following command in the terminal:

```
$ python kamins.py victimIP targetZone yourNameserverIP [-t/--ttl ttl]
      [-s/--silent -l/--loud] [-d/--demo authoritativeNameserverIP]
```

Arguments without the dash (`-`) in front of them are positional, meaning that they are required and need to be positioned exactly in the ordering described above. Arguments with a dash (`-`) in front of them are optional and may be ordered in any way desired, as long as they are after the positional arguments. Each of the given arguments is described below:

- `victimIP` - this argument represents the IP of the victim nameserver, whose cache is supposed to be poisoned after attack execution.
- `targetZone` - this arguments represents the zone that is going to be spoofed after the attack is executed.
- `yourNameserverIP` - this argument represents the IP of the nameserver that the attacker wishes to become authoritative for the target zone, with respect to the victim.
- `-t/--ttl ttl` - the value after the positional argument is going to be the Time-To-Live of the record in the victims cache.
- `-s/--silent` - this argument specifies that the attacker wishes to execute the silent variant of the attack. If neither this nor the following argument are provided, silent method will be executed by default.
- `-l/--loud` - this argument specifies that the attacker wishes to execute the loud variant of the attack. It is important to note this argument are the one described above are exclusive, meaning that at most one can be provided.
- `-d/--demo authoritativeNameserverIP` - this is the IP of the authoritative nameserver used in the demo. It is needed because normal NS record lookup for the target zone won't provide this IP, as the demo is executed in a closed environment.

Attack analysis

As stated in the first part of the report, this tool implements two methods, silent and loud. Silent method is completely anonymous, apart from the ARP cache poisoning of the victim nameserver. In this case authoritative nameserver's IP is associated with the attacker's MAC address. It should be noted that in the case that the attacker has a foothold in the local network, it is not his MAC address that is associated with the authoritative nameserver's IP, but that of the host that is executing this script. However, this method requires that the victim nameserver, authoritative nameserver and the host executing this script be in the same local network. This is because the Address Resolution Protocol (ARP) operates only in local networks. On the other hand, loud method does not have this constraint. However, its operation generates a lot of noise, since the victim is flooded with fake response packets trying to match the pending query's TXID. Both of the methods start in the same manner, hence their common start is explained first.

3.1 Common start

Firstly, the script validates input in order to verify that the supplied arguments are valid. Afterwards, it fetches the NS records for the target domain, and their respective IPs. This information is is used later to forge the appropriate packets, and in the case of the silent method, ARP cache poison the victim nameserver. Finally, the script checks whether all the requirements described below are fulfilled before executing the actual attack. The victim is queried for the IP of the target zone supplied in the arguments. This is done for the following reasons:

- Victim needs to be a nameserver responding to queries in order for this attack to be executed.
- Victim cannot be authoritative for the target zone, because an authoritative nameserver cannot be cache poisoned for the zone that it is authoritative for.
- Victim needs to facilitate recursive query resolution, because non-recursive nameserver's cannot be cache poisoned with this attack.

If all of the above conditions are met, the script proceeds to executing the actual attack, depending on the method supplied in the arguments.

3.2 Loud method

This method iterates over a loop until the attack is successfully executed. The first step is generating a random subdomain that the victim is going to be queried upon. The query request is then sent to the victim. Since this subdomain is randomly generated, we assume that it is not cached in the victim's memory. Hence, the victim will proceed with the normal recursive query resolution, eventually querying the authoritative nameserver for the IP of the randomly generated subdomain. It is worth mentioning that the recursive query resolution will start from root servers only on the first try, since afterwards the victim will cache the authoritative nameserver's IP and the appropriate NS record.

It is important to note that the recursive nameserver will accept a response to a query only in the case that the TXID matches that of the requests. The TXID is a 16-bit number, so there are 65536 possible values. This method is trying to match the TXID in the forged response before the authoritative nameserver answers, and the query gets resolved.
When the query is sent to the victim, the script starts flooding the victim with forged ”response” packets, spoofing the authoritative nameserver. The response packets have the following structure:

- Question section of the DNS header contains the query for the previously generated random subdomain.
- Answer RR section of the DNS header is empty.
- Authority RR section contains a single NS record, stating that the authoritative nameserver for the target zone is the actual authoritative nameserver, which the script got in the first steps of the execution, explained in the previous section.
- Additional RR section contains a single A record, stating that the authoritative nameserver's IP is the IP to be spoofed, provided in the arguments.

Last item on the list is the key of the attack, since the victim is going to overwrite the authoritative nameserver's A record stored in the cache, with the record supplied in the ”glue” of the response.
After the flood of responses is sent, the victim is queried for the A record of the authoritative nameserver, whose domain was obtained in the first steps of the execution, explained in section 3.1. If the response matches the IP provided in the arguments, the victim was successfully cache poisoned, and the script terminates. Otherwise, the whole process described in this section is repeated.
The downsides of this approach are obviously the noise that it generates. Moreover, the time that it will take for the attack to be successful is completely non-deterministic since the TXIDs may never match. Finally, since this vulnerability was discovered, source port randomization for queries has been implemented which makes the guessing even harder since the port is an 11-bit number (because of reserved ports), hence effectively making the guess in the 27-bit range, which has approximately 134 million possible values. Our script does not leverage for source port randomization, as it always sends queries from port 53. This could be an improvement for the future iteration of the script.

3.3 Silent method

This method is a bit less naive than the loud method described above, since it does not rely on luck to successfully execute the attack. However, as mentioned earlier, it assumes that all the significants hosts are in a local network.
The attack starts by ARP poisoning both the victim and the authoritative nameservers, redirecting all the traffic to the attacker. This is done by sending a gratuitous ARP packet associating the other hosts IP with the attacker's MAC address. Next, a random subdomain is generated, and the victim is queried for its IP. Scapy then starts sniffing for packets, looking for a query from the victim to the authoritative nameserver asking the same question, and forwarding all other packets. We are sure that the request is going to pass through the

attacker, because of the previous ARP cache poisoning step. When such a packet arrives, script forges and sends a new response packet with the following characteristics:

- IP source and destination are switched with respect to the request, in order to simulate the response from the authoritative nameserver to the victim.
- UDP source and destination port are switched with respect to the request, in order to mitigate response dropping due to source port randomization for DNS queries.
- TXID in the DNS header is copied for the request, in order to simulate a valid response.
- Authoritative answer flag for the DNS header is set to true.
- Question section of the DNS header contains the query for the previously generated random subdomain.
- Answer RR section of the DNS header is empty.
- Authority RR section contains a single NS record, stating that the authoritative nameserver for the target zone is the actual authoritative nameserver, which the script got in the first steps of the execution, explained in section 3.1.
- Additional RR section contains a single A record, stating that the authoritative nameserver's IP is the IP to be spoofed, provided in the arguments.

Since the request never reaches the authoritative nameserver, since we do not forward it, we are sure that the authoritative nameserver is not going to respond to the victim. Therefore, we do not have to worry about winning the race.
As in the loud method, additional RR section of the DNS header is the key of this attack, since this is the record that will actually poison the victim.
Unlike the loud method, this method does not require the source port to be 53, hence making it more appropriate for modern scenarios.

Technical setup

To produce this attack we used multiple virtual machines in VirtualBox running Debian9 with the network-setting put to host-only adapter.

The first machine is meant for the attacker. This machine has Scapy, Python and git installed. The second machine is meant for the recursive DNS server (the victim machine). This machine has bind9 installed to handle the DNS-requests, and wireshark to analyze the traffic. The machine has a specific setup to allow recursion, to not use any of the new dnssec implementation and to forward it's requests to our own authoritative DNS-server. This authoritative DNS-server we did not actually make as we ARP-spoof this machine in the beginning to our second machine so it never actually has to respond to/do anything.

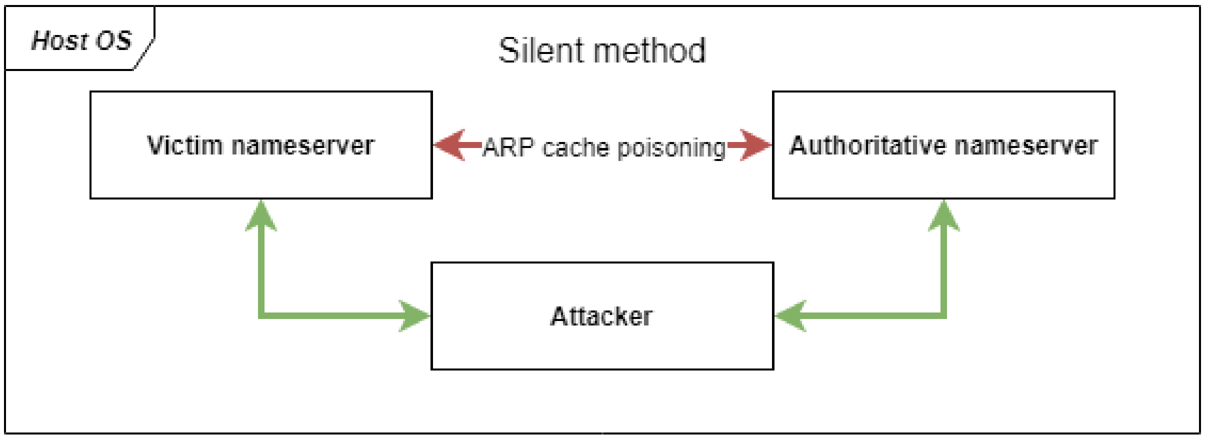

Figure 1: Network topology for the silent method

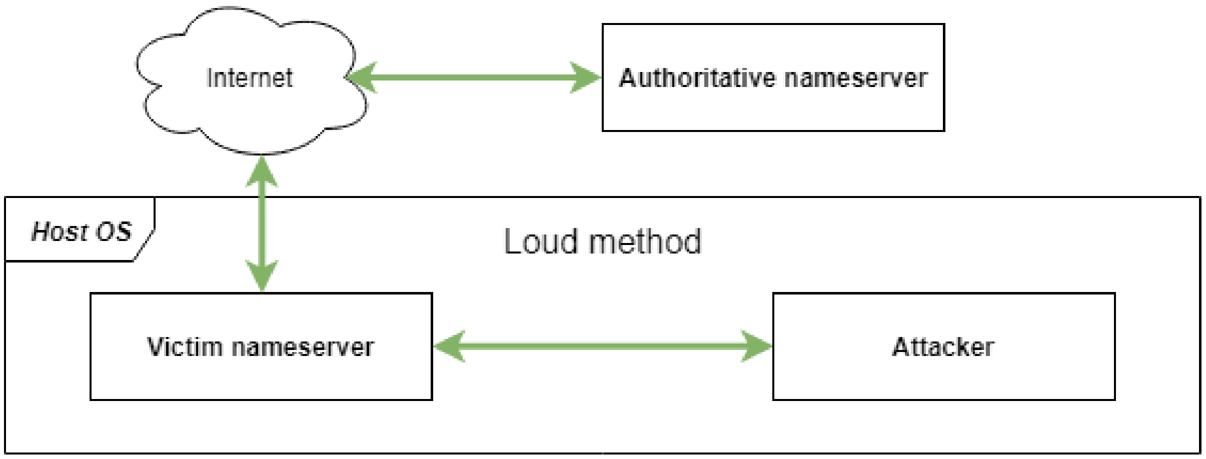

Figure 2: Network topology for the loud method

Source code

Source code can be downloaded from the publicly available GitHub repository, on this URL: `https://github.com/filipdavidovic/kaminsky_vulnerability`.