DNS Cache Poisoning Assignment

Attack Introduction and Assignment Description

The Domain Name System (DNS) is a hierarchical distributed database which together with the homonym application-level protocol (DNS), allows every Internet host to “translate” the more memorisable domain names into IP addresses. However, the DNS provides other features which are worth to mention but not really needed to complete/understand the assignment and the exploited vulnerability. Most important among them are known as host aliasing, which allows hosts with complicated hostname to have one or more alias names, and load distribution, which briefly means more IP addresses can be associated to the same hostname.

Usually, when a client decides to obtain the IP address of a specific domain name, it sends out a DNS query containing the interested domain names to its default DNS server (i.e. ISP’s DNS server in a typical home Internet connections). The DNS server that receive the query may be the authoritative server for the requested domain(s), and then answer with proper resource records, or it may be not authoritative and then it would need to inform the client he’s not the answers he was looking for. In the first scenario, the server just answer with (as an example) the IP addresses requested. On the other hand, if it has not the answers:

1. (recursive DNS) it either tries to find them by asking to other DNS servers the same questions, storing obtained information in its cache in the meanwhile, or
2. it answers with the IP addresses *the client* should contact in order to obtain what he was looking for

In 2008, during a BlackHat conference[1][2], Dan Kaminsky presented a Domain Name System (DNS) vulnerability that can allow attackers to redirect network clients to alternative server of their own choosing. This can be done by inserting bogus entries into a recursive nameserver’s cache, thus implying that every client that requests the affected domain will be redirected to a different malicious server without notice anything. The vulnerability explained was due to a non-safe use of the queryID and UDP source port used for sending requests.

The query ID is a unique identifier created in the query packets that’s left intact by the server sending the reply and it’s used to allow the server making the request to associate the answers with the correct questions.

However, in older name servers, it was only 16 bits long and it was either increased by 1 for every new query, or generated by poor pseudo-random number generators. Additionally, since name servers were usual to send out requests with the same UDP source port, an attacker knowing the DNS server that a specific target is going to contact, the source port of the request, and by guessing the next

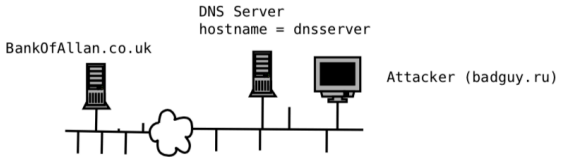
---

[1] https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf

[2] https://www.youtube.com/watch?v=7Pp72gUYx00

queryID that will be used, he would be able to poison the target's cache by sending spoofed DNS answers each with a different query ID.

The goal of the assignment is to implement a script which poison the cache of a recursive DNS server so that all requests for the IP address of BankOfAllan return an IP address of the attacker's choice. The assignment scenario is depicted in the following image.



The attacker owns the domain badguy.ru, which can be used to obtain the source UDP port and queryID used by the DNS server for sending recursive queries.

The DNS server has two interfaces: one reachable by the attacker and the other that is used for sending out recursive queries.

On successful poisoning of the DNS cache, a unique secret will be sent as a UDP packet to port 1337.

Required Information and Assumptions

Before explaining how I gathered the required information and how the developed script works, is important to remember the assumptions made. I assumed that:

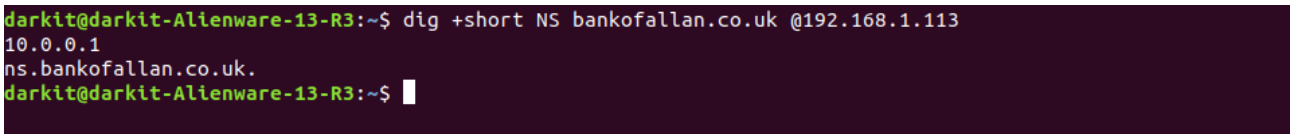
- the script is run on a *nix-like system;
- the command-line utility **`dig`** is installed and working properly on the host where the script is run; and
- a Python 2.7 interpreter and the latest version of the Scapy library have been both correctly installed on the host where the script is run.

Below is a list of the required data (IPs and ports) for starting a cache poisoning attack together with a brief description of the technique I used to obtain them.

- The IP address corresponding to the interface used by the target server (**192.168.1.113**) for receiving DNS queries. This is the address that will be used during the attack as the destination address of the outgoing spoofed datagrams. In any realistic scenario this address have to be known to the attacker. In other words, it can even be retrieved by using the `ifconfig` utility, but that would mean you've already obtained remote access to the host

running the server, thus I assumed it is known to the attacker.

- The IP address of the authoritative server for the bankofallan.co.uk domain (**10.0.0.1**), which is the one that will be used as (spoofed) source address into every packet sent as part of the flooding step - the phase of the attack where the attacker floods the target with several DNS query answers with the aim to win the race against a legitimate authoritative server which is supposed to send the same answer at almost the same time. In order to obtain this IP address I used the `dig` utility as shown into the following screenshot.



- The current UDP source port and queryID used by the target DNS server for sending recursive queries. They're necessary (together with the information above) in order to fool the vulnerable server, otherwise it would reject DNS answers with non-matching source port and queryID. They can be obtained by running a simple Python script which acts as the authoritative nameserver for the badguy.ru domain and saves the received (interesting) values, for example, into a text file. Once this "DNS server" is running, we can just query the vulnerable server for the domain we own, thus obtaining those values. Here's a snippet of the code used into the exploit script for performing what's just been mentioned.

```
def dns():
  sock = socket(AF_INET, SOCK_DGRAM)
  sock.bind(('0.0.0.0', 53))
  print "[*] DNS SERVER STARTED\n"
  while poisoned != 1:
      request, addr = sock.recvfrom(4096)
     udp_sourceport = addr[1]
      try:
          dns = DNS(request)
          queryID = dns.id
          assert dnsqtypes[dns[DNSQR].qtype] == 'A', dns[DNSQR].qtype
          query = dns[DNSQR].qname.decode('ascii')
          if query[-1] == '.':
              query = query[:-1]
         assert query == badguy_domain

          response = DNS(
              id=dns.id, ancount=1, qr=1, rd=0, aa=1,
              an = DNSRR(rrname=str(query), type='A', rdata=rogueIP, ttl=3600)
              )
          sock.sendto(bytes(response), addr)
      except Exception as e:
          print "Error during DNS query handling"

    # writes the tuple (UDPport,queryID) into a regular file
      with open(filename, 'w') as f:
          f.write(str(udp_sourceport)+','+str(queryID))
```

To sum up, the required data:

- IP address of the vulnerable DNS server — **192.168.1.113** in my environment
- IP address to insert into the DNS's cache — **192.168.1.109**, attacker's choice
- IP address of the authoritative NS for the domain bankofallan.co.uk, obtained using the dig utility — **10.0.0.1**
- Current queryID and UDP source port (both detected and repeatedly updated at run-time) obtained by asking to the vulnerable server "what is the IP of badguy.ru?"
- Port 53 hasn't been mentioned because should be the obvious missing port. It is the default port used for DNS communications and the vulnerable server is listening on port 53, thus every forged datagram sent during the flood need to have "53" as source UDP port for being accepted by the vulnerable server and, additionally, attacker's DNS server needs to listen on that UDP port.

How the script works

The exploit is a command-line Python script which takes two IP addresses as arguments.

```
Usage: python exploit.py {target_dns} {attacker_ip}
```

First argument is self-explanatory; second argument represents the address the attacker wants to inject into the server's cache.

When it is executed, the script spawns 3 threads: (1) the authoritative DNS server for the badguy.ru domain name, which is used for obtaining/updating the current query ID and UDP source port used by the vulnerable server; (2) the listener on UDP port 1337, which is supposed to intercept the secret; and (3) a thread which tries to win the race by sending DNS answers with forged source IP and query ID for the domain that the vulnerable server is trying to recursively resolve. The thread that floods the target, updates the needed parameters every *n* packets sent (configurable by modifying lines of the script).

In order to poison the cache, just run the exploit with correct parameters and then execute the following "infinite" loop into another terminal window.

```
while true; do dig random.bankofallan.co.uk @TARGET_DNS;done;
```

This loop is used to force the vulnerable sever to make recursive queries, which allows the thread (3) to inject an arbitrary IP address for the requested domain (random.bankofallan.co.uk) if the query ID guessed correctly.
When the cache is poisoned successfully, the thread (2) prints the received secret and the attack is terminated.

Note: both the DNS server and the listener bind on all the interfaces.

Concurrency Considerations

The exploit script spawns several thread and two of them need to access the same resource: the text file. It is accessed by both the DNS and packets flooder threads. However, unless the DNS server for the bad guy domain receives other unexpected queries, there is no way the reading thread will read inconsistent data: this because the relevant data are never read and write at the same time.

If the DNS server owned by the attacker receives other queries (those that are not sent by the exploit script itself), then concurrency should be taken into account for avoiding the creation of inconsistent data and other issues. The latter scenario hasn't been taken into consideration because was not the goal of the assignment.

Why Python and Scapy?

Initially I thought to write the code using the C programming language but, when I realised that a Python script would have been cleaner (and easier to write) once written, I decided to use it. Ruby and Java had also been considered but I preferred Python due to some more experience and libraries support. Additionally, by using Python I didn't have to manage memory on my own and I had the opportunity to use the powerful Scapy library. Other packet-manipulation libraries have been taken into account (e.g. the `dnslib` library) but Scapy won due to its speed, easy of use, and available documentation.

Results and Configuration

The exploit worked without any modification every time it's been tested.
Using the following config.json:

```
root@osboxes:~# cat config.json
{"badguyIP":"192.168.1.109","secret": "a231bc78c0f8", "queryPort": 31337}
root@osboxes:~#
```

I obtained the following secret:

```
darkit@darkit-Alienware-13-R3:~/Desktop/New-trying$ sudo python exploit.py 192.168.1.113 192.168.1.109
[*] DNS SERVER STARTED
[*] LISTENER STARTED ON UDP PORT 1337

[*] Flooder started
[*] flooder is updating the current queryID...
[*] flooder is updating the current queryID...
[*] flooder is updating the current queryID...


SECRET RECEIVED!!!!!


038cbef5d8499830cf9a9d2670ab3e66

[*] flooder is updating the current queryID...
[*] Attack succeeded!
darkit@darkit-Alienware-13-R3:~/Desktop/New-trying$
```

Instead, using the following configuration file:

```
root@osboxes:~# cat config.json
{"badguyIP":"192.168.1.109","secret": "12345zzzzzzz", "queryPort": 31337}
root@osboxes:~#
```

I obtained:

```
[*] DNS SERVER STARTED

[*] LISTENER STARTED ON UDP PORT 1337
[*] Flooder started
[*] flooder is updating the current queryID...
[*] flooder is updating the current queryID...
[*] flooder is updating the current queryID...


SECRET RECEIVED!!!!!


5c34b827af21b017806ef7179420be71

[*] Attack succeeded!
darkit@darkit-Alienware-13-R3:~/Desktop/New-trying$
```