Lab:4 Secure E-Mail and DNS

Secure computer networks D7075E

Group 35

Vatousiadi Spyridoula – spyvat-5@student.ltu.se
Stefanos Ntentopoulos – stente-5@student.ltu.se

1 November 2025

1 Introduction

1.1 Objectives

This report documents the implementation and analysis of email and DNS security mechanisms in a containerized network environment. The lab demonstrates how to build a realistic ISP/-domain network topology with mail and DNS infrastructure, demonstrate vulnerabilities in unsecured email and DNS systems, and how to implement security mechanisms: SPF, DKIM, DMARC, and DNSSEC. Finally, we will validate security improvements through testing and traffic analysis.

1.2 Lab Environment

The project began with the setting up the topology of our lab. The lab environment consists of four Docker containers:

- **DNS Server** (172.20.0.10): BIND9 authoritative nameserver. This container acts as the authoritative DNS server for the example.com domain using BIND. Its primary role is to provide name resolution (A records, MX records) and, later, to publish critical security records like SPF, DKIM, and DMARC, and to implement DNSSEC for zone signing.
- **Mail Server** (172.20.0.20): Postfix SMTP server with OpenDKIM. This container functions as the mail exchange server using Postfix and OpenDKIM. It handles the acceptance and processing of emails for user@example.com on standard ports (25, 587, 993). This server is used to demonstrate both the initial vulnerability to header forgery and the later mitigation provided by DKIM signing and SPF checks.
- **Client** (172.20.0.30): User workstation for testing. This container simulates the host machine or a user's email client. It is the primary testing point, used to query the DNS server and send emails via swaks. It serves as the target (or victim) for the DNS spoofing attack and the source for testing all security protocols.
- **Attacker** (172.20.0.40): Adversarial host for attack simulations. his container is the adversarial machine, hosting the necessary scripts (DNS spoofer and fake mail server) to execute the lab's required attacks, specifically targeting the client's DNS lookups and attempting to intercept mail traffic.

All containers operate on an isolated Docker network 4lab_labnet (172.20.0.0/16) to provide a controlled testing environment.

```
CONTAINER ID   IMAGE              STATUS          NAMES
b32ba4df03b2   4lab-client        Up 25 minutes   client
09dcc94bd2ca   4lab-mail-server   Up 25 minutes   mail-server
45800c8548ed   4lab-dns-server    Up 23 minutes   dns-server
d32a1703b21e   4lab-attacker      Up 32 minutes   attacker
steven@steven-VirtualBox:~/Desktop/4Lab$ docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
451c2e1d35f4   4lab_labnet   bridge    local
bba85bb11f5d   bridge        bridge    local
241b706cff24   host          host      local
098d529b3a18   none          null      local
```

2 Theoretical Background

2.1 DNS Security

2.1.1 DNS Fundamentals

The Domain Name System translates human-readable domain names into IP addresses. Key DNS record types include:

- **A Records**: Map domain names to IPv4 addresses
- **MX Records**: Specify mail exchange servers for a domain
- **NS Records**: Identify authoritative nameservers
- **TXT Records**: Store arbitrary text data (used for SPF, DKIM, DMARC)

2.1.2 DNS Vulnerabilities

Traditional DNS is vulnerable to:

- **Cache Poisoning**: Injecting false DNS records into resolver caches
- **Spoofing**: Forging DNS responses to redirect traffic
- **Man-in-the-Middle**: Intercepting and modifying DNS queries/responses

2.1.3 DNSSEC Protection

DNS Security Extensions (DNSSEC) provide cryptographic authentication through:

- **Digital Signatures**: Each DNS record is signed with a private key
- **Chain of Trust**: Hierarchical validation from root to authoritative server
- **RRSIG Records**: Resource Record Signatures prove data authenticity
- **DNSKEY Records**: Public keys for signature verification
- **DS Records**: Delegation Signer records link parent and child zones

DNSSEC uses two types of keys:

- **Zone Signing Key (ZSK)**: Signs individual DNS records (2048-bit)
- **Key Signing Key (KSK)**: Signs the DNSKEY record set (4096-bit)

2.2 Email Security

2.2.1 SMTP Vulnerabilities

The Simple Mail Transfer Protocol has inherent security weaknesses:

- No sender authentication
- Headers can be trivially forged
- No message integrity verification
- Plain text transmission (without TLS)

2.2.2 SPF (Sender Policy Framework)

SPF allows domain owners to specify which mail servers are authorized to send email on their behalf.

How SPF Works:

1. Domain publishes SPF record in DNS as TXT record
2. Receiving server checks sender IP against SPF policy
3. Result: Pass, Fail, SoftFail, Neutral, or PermError

Example SPF Record:

```
example.com. IN TXT "v=spf1 ip4:172.20.0.20 -all"
```

This record authorizes only 172.20.0.20 to send mail for example.com, with `-all` indicating strict rejection of unauthorized senders.

2.2.3 DKIM (DomainKeys Identified Mail)

DKIM provides cryptographic proof that an email was authorized by the domain owner.

DKIM Process:

1. Sending server signs email headers and body with private key
2. Signature is added as DKIM-Signature header
3. Public key is published in DNS
4. Receiving server retrieves public key and verifies signature

DKIM Signature Components:

- **v**: Version (DKIM1)
- **a**: Algorithm (rsa-sha256)

- **d**: Signing domain
- **s**: Selector (identifies specific key)
- **h**: Signed headers
- **b**: Signature value (base64-encoded)

2.2.4 DMARC (Domain-based Message Authentication, Reporting & Conformance)

DMARC builds on SPF and DKIM to provide policy enforcement and reporting.

DMARC Policy Options:

- **none**: Monitor only, no enforcement
- **quarantine**: Move suspicious emails to spam
- **reject**: Reject emails that fail authentication

Example DMARC Record:

```
_dmarc.example.com. IN TXT "v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com"
```

DMARC Benefits:

- Coordinates SPF and DKIM results
- Provides aggregate reporting on authentication failures
- Allows gradual policy enforcement
- Protects brand reputation

3 Implementation

3.1 Initial Setup

3.1.1 Environment Deployment

The laboratory environment was deployed using Docker Compose with the following architecture:

```
docker compose up -d --build
```

3.1.2 DNS Functionality Verification

DNS resolution was tested for all critical record types:

A Record Query:

```
dig @172.20.0.10 mail.example.com
```

Expected result: mail.example.com resolves to 172.20.0.20

```
root@client:/# dig @172.20.0.10 mail.example.com

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @172.20.0.10 mail.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58332
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 627fb6a03b7a365d0100000069073860542528cf35228f93 (good)
;; QUESTION SECTION:
;mail.example.com.              IN      A

;; ANSWER SECTION:
mail.example.com.       604800  IN      A       172.20.0.20

;; Query time: 4 msec
;; SERVER: 172.20.0.10#53(172.20.0.10) (UDP)
;; WHEN: Sun Nov 02 10:54:24 UTC 2025
;; MSG SIZE  rcvd: 89
```

MX Record Query:

```
dig @172.20.0.10 example.com MX
```

Expected result: MX record pointing to mail.example.com with priority 10

```
root@client:/# dig @172.20.0.10 example.com MX

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @172.20.0.10 example.com MX
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63787
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 02d311945999aa52010000069073900e0c9b5bd2cfa6695 (good)
;; QUESTION SECTION:
;example.com.                   IN      MX

;; ANSWER SECTION:
example.com.            604800  IN      MX      10 mail.example.com.

;; ADDITIONAL SECTION:
mail.example.com.       604800  IN      A       172.20.0.20

;; Query time: 0 msec
;; SERVER: 172.20.0.10#53(172.20.0.10) (UDP)
;; WHEN: Sun Nov 02 10:57:04 UTC 2025
;; MSG SIZE  rcvd: 105
```

NS Record Query:

```
dig @172.20.0.10 example.com NS
```

```
root@client:/# dig @172.20.0.10 example.com NS

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @172.20.0.10 example.com NS
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64915
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 579b55cdbf56a30301000000069073988fc133d5214eefe1c (good)
;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            604800  IN      NS      dns.example.com.

;; ADDITIONAL SECTION:
dns.example.com.        604800  IN      A       172.20.0.10

;; Query time: 2 msec
;; SERVER: 172.20.0.10#53(172.20.0.10) (UDP)
;; WHEN: Sun Nov 02 10:59:20 UTC 2025
;; MSG SIZE  rcvd: 102
```

3.1.3 Email Delivery Testing

Initial email functionality was verified using SWAKS (Swiss Army Knife for SMTP):

```
swaks --to user@example.com --from testuser@client.example.com --server
    mail.example.com --header "Subject: Test Email 1" --body "This is a test email
    from the client."
```

```
root@client:/# swaks --to user@example.com --from testuser@client.example.com --server mail.example.com --header "Subject: Test Email 1" --body "This
   is a test email from the client."
=== Trying mail.example.com:25...
=== Connected to mail.example.com.
<-  220 mail.example.com ESMTP
 -> EHLO client.example.com
<-  250-mail.example.com
<-  250-PIPELINING
<-  250-SIZE 10240000
<-  250-VRFY
<-  250-ETRN
<-  250-ENHANCEDSTATUSCODES
<-  250-8BITMIME
<-  250-DSN
<-  250 CHUNKING
 -> MAIL FROM:<testuser@client.example.com>
<-  250 2.1.0 Ok
 -> RCPT TO:<user@example.com>
<-  250 2.1.5 Ok
 -> DATA
<-  354 End data with <CR><LF>.<CR><LF>
 -> Date: Sun, 02 Nov 2025 13:48:44 +0000
 -> To: user@example.com
 -> From: testuser@client.example.com
 -> Subject: Test Email 1
 -> Message-Id: <20251102134844.000083@client.example.com>
 -> X-Mailer: swaks v20201014.0 jetmore.org/john/code/swaks/
 ->
 -> This
 ->    is a test email from the client.
 ->
 ->
 -> .
<-  250 2.0.0 Ok: queued as 07E55C41C83
 -> QUIT
<-  221 2.0.0 Bye
=== Connection closed with remote host.
```

The email was successfully queued and delivered, confirming baseline SMTP functionality.

```
Nov  2 13:52:01 mail postfix/smtpd[64]: warning: dict_nis_init: NIS domain name not set - NIS lookups disabled
Nov  2 13:52:01 mail postfix/smtpd[64]: connect from unknown[172.20.0.30]
Nov  2 13:52:01 mail postfix/smtpd[64]: 5D7BAC41C87: client=unknown[172.20.0.30]
Nov  2 13:52:01 mail postfix/cleanup[67]: 5D7BAC41C87: message-id=<20251102135201.000084@client.example.com>
Nov  2 13:52:01 mail postfix/qmgr[28]: 5D7BAC41C87: from=<testuser@client.example.com>, size=464, nrcpt=1 (queue active)
Nov  2 13:52:01 mail postfix/smtpd[64]: disconnect from unknown[172.20.0.30] ehlo=1 mail=1 rcpt=1 data=1 quit=1 commands=5
Nov  2 13:52:01 mail postfix/local[68]: warning: dict_nis_init: NIS domain name not set - NIS lookups disabled
Nov  2 13:52:01 mail postfix/local[68]: 5D7BAC41C87: to=<user@example.com>, relay=local, delay=0.04, delays=0.02/0.01/0/0.01, dsn=5.2.0, status=bounced (cannot update mailbox /var/mail/user for user user. cannot open file: Is a directo
ry)
Nov  2 13:52:01 mail postfix/cleanup[67]: 658BDC41C89: message-id=<20251102135201.658BDC41C89@mail.example.com>
Nov  2 13:52:01 mail postfix/bounce[69]: 5D7BAC41C87: sender non-delivery notification: 658BDC41C89
Nov  2 13:52:01 mail postfix/qmgr[28]: 658BDC41C89: from=◇, size=2388, nrcpt=1 (queue active)
Nov  2 13:52:01 mail postfix/qmgr[28]: 5D7BAC41C87: removed
Nov  2 13:52:01 mail postfix/smtp[70]: 658BDC41C89: to=<testuser@client.example.com>, relay=none, delay=0.01, delays=0/0.01/0/0, dsn=4.4.3, status=deferred (Host or domain name not found. Name service error for name=client.example.com
type=MX: Host not found, try again)
Nov  2 13:52:01 mail postfix/smtp[70]: using backwards-compatible default setting relay_domains=$mydestination to update fast-flush logfile for domain "client.example.com"
```

3.2 Demonstration of Vulnerabilities

3.2.1 Email Header Forgery Attack

With no authentication mechanisms in place, email headers were trivially forged:

```
swaks --to victim@example.com \
      --from ceo@example.com \
      --header "From: CEO <ceo@example.com>" \
      --server mail.example.com \
      --header "Subject: URGENT: Wire Transfer Required" \
```

```
    --body "Please wire $50,000 to account 123456 immediately."
```

Result: As we cam see the forged email was accepted and delivered without any validation.

```
└─# docker exec mail-server cat /var/mail/victim
From ceo@example.com  Sun Nov  2 13:59:41 2025
Return-Path: <ceo@example.com>
X-Original-To: victim@example.com
Delivered-To: victim@example.com
Received: from client.example.com (unknown [172.20.0.30])
        by mail.example.com (Postfix) with ESMTP id 42224C41C5B
        for <victim@example.com>; Sun,  2 Nov 2025 13:59:41 +0000 (UTC)
Date: Sun, 02 Nov 2025 13:59:41 +0000
To: victim@example.com
From: CEO <ceo@example.com>
Subject: URGENT: Wire Transfer Required
Message-Id: <20251102135941.000087@client.example.com>
X-Mailer: swaks v20201014.0 jetmore.org/john/code/swaks/

Please wire 0,000 to account 123456 immediately.
```

Analysis: This demonstrates a critical vulnerability. An attacker can impersonate any user, including executives, to conduct various phishing attacks.

3.2.2 DNS Spoofing Attack

In additions a DNS spoofing attack was simulated using the attacker container:

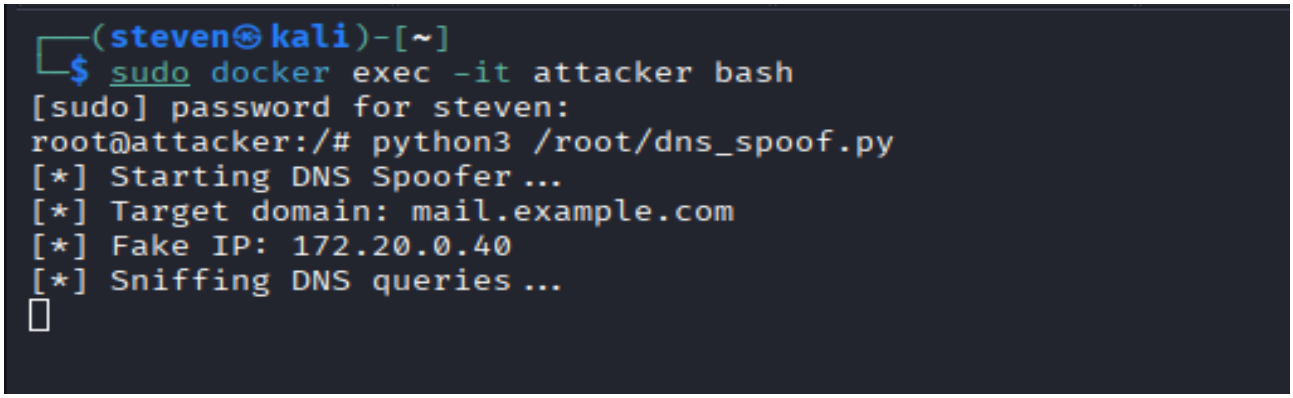
Terminal 1 - Fake Mail Server:

```
python3 /root/fake_mail_server.py
```

```
root@attacker:/# python3 /root/fake_mail_server.py
[*] Fake SMTP server listening on 0.0.0.0:25
[*] Waiting for connections...
```

Terminal 2 - DNS Spoofer:

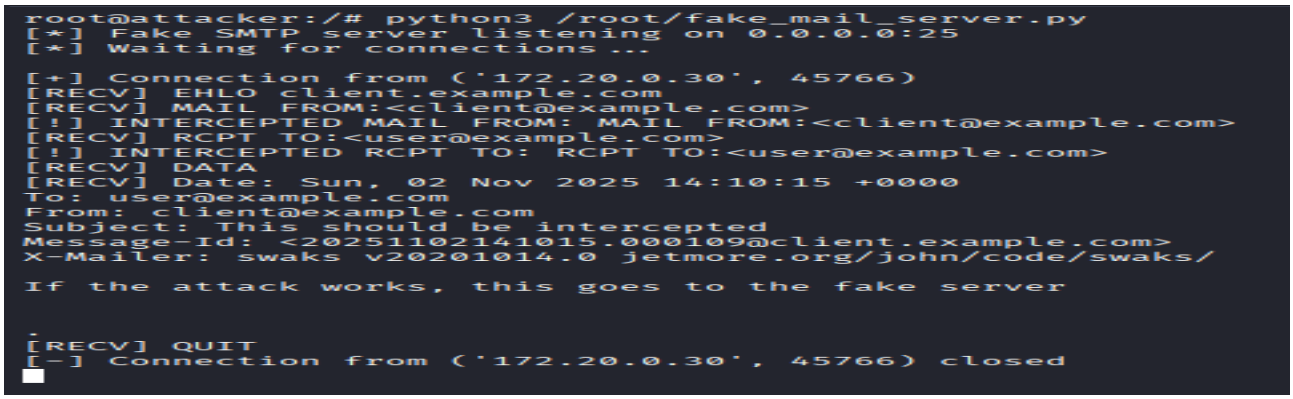
```
python3 /root/fake_dns_server.py
```



Client Configuration:

```
echo "nameserver 172.20.0.40" > /etc/resolv.conf
```

When the client was configured to use the attacker's DNS server, all email traffic was redirected to the fake mail server at 172.20.0.40.



```
root@attacker:/# python3 /root/dns_spoof.py
[*] Starting DNS Spoofer ...
[*] Target domain: mail.example.com
[*] Fake IP: 172.20.0.40
[*] Sniffing DNS queries ...
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.
[+] Sent spoofed response: mail.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
[+] Spoofing DNS response for mail.example.com.example.com.
[+] Sent spoofed response: mail.example.com.example.com. → 172.20.0.40
```

```
root@attacker:/# python3 /root/fake_dns_server.py
[*] Starting Fake DNS Server ...
[*] Target domain: mail.example.com
[*] Spoofed IP: 172.20.0.40
[*] Real DNS server: 172.20.0.10
[*] Listening on UDP port 53 ...

[*] Received query from 172.20.0.30 for: example.com
[*] Forwarding to real DNS server ...
[*] Received query from 172.20.0.30 for: example.com
[*] Forwarding to real DNS server ...
[*] Received query from 172.20.0.30 for: mail.example.com
[!] SPOOFING: mail.example.com → 172.20.0.40
```

Analysis: Without DNSSEC, clients cannot verify the authenticity of DNS responses, allowing attackers to redirect traffic to malicious servers.

3.3 SPF Implementation

3.3.1 SPF Record Configuration

An SPF record was added to the DNS zone to authorize the mail server:

DNS Zone File Addition:

```
@ IN TXT "v=spf1 ip4:172.20.0.20 -all"
```

Serial number was incremented (2023110201 → 2023110202) to trigger zone reload.

DNS Reload:

```
kill -HUP $(cat /var/run/named/named.pid)
```

3.3.2 SPF Verification

```
dig @172.20.0.10 example.com TXT
```

```
root@client:/# dig @172.20.0.10 example.com TXT

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @172.20.0.10 example.com TXT
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6304
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 580e110fcd12c48f0100000069079b8b7646b84e224ecf1a (good)
;; QUESTION SECTION:
;example.com.                   IN      TXT

;; ANSWER SECTION:
example.com.            604800  IN      TXT     "v=spf1 ip4:172.20.0.20 -all"

;; Query time: 0 msec
;; SERVER: 172.20.0.10#53(172.20.0.10) (UDP)
;; WHEN: Sun Nov 02 17:57:31 UTC 2025
;; MSG SIZE  rcvd: 108

root@client:/#
```

The SPF record was successfully published and resolving correctly. This SPF record states that only the mail server at 172.20.0.20 is authorized to send email for the example.com domain.

SPF Policy Interpretation:

- **v=spf1**: SPF version 1
- **ip4:172.20.0.20**: Authorize this IP to send mail
- **-all**: Hard fail for all other sources

3.4 Phase 4: DKIM Implementation

3.4.1 DKIM Key Generation

DKIM keys were automatically generated during mail server initialization:

```
opendkim-genkey -b 2048 -d example.com -s default -v
```

```
≡ DKIM Key Generation Verification ≡

OpenDKIM Process Status:
opendkim     16  0.0  0.0  22280  2628 ?        Ss   14:49   0:00 /usr/sbin/opendkim -u opendkim -p inet:8891@localhost
opendkim     17  0.0  0.0 260132  9296 ?        Sl   14:49   0:00 /usr/sbin/opendkim -u opendkim -p inet:8891@localhost

DKIM Keys Generated:
total 8.0K
-rw------- 1 opendkim opendkim 1.7K Nov  2 12:36 default.private
-rw------- 1 opendkim opendkim  507 Nov  2 12:36 default.txt

OpenDKIM Startup Log:
opendkim     16  0.0  0.0  22280  2680 ?        Ss   14:48   0:00 /usr/sbin/opendkim -u opendkim -p inet:8891@localhost
opendkim     17  0.0  0.0 129060  6948 ?        Sl   14:48   0:00 /usr/sbin/opendkim -u opendkim -p inet:8891@localhost
Starting OpenDKIM...
opendkim     16  0.0  0.0  22280  2628 ?        Ss   14:49   0:00 /usr/sbin/opendkim -u opendkim -p inet:8891@localhost
opendkim     17  0.0  0.0 129060  7108 ?        Sl   14:49   0:00 /usr/sbin/opendkim -u opendkim -p inet:8891@localhost
```

Keys generated:

- Private key: /etc/opendkim/keys/example.com/default.private (2048-bit RSA)
- Public key: /etc/opendkim/keys/example.com/default.txt

3.4.2 DKIM DNS Record

The DKIM public key was published in DNS:

```
default._domainkey IN TXT ( "v=DKIM1; h=sha256; k=rsa; "
    "p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAh7Td7VIuQwb7t..."
    "Wna5S/iMD4wdXjf24aYpoY53nR9fCAPXwgYwnRs5OFIfK7wnkd0SIyaEPj..."
    "...IDAQAB" )
```

3.4.3 OpenDKIM Integration

Postfix was configured to sign outgoing emails via OpenDKIM:

main.cf Configuration:

```
milter_default_action = accept
milter_protocol = 2
smtpd_milters = inet:127.0.0.1:8891
non_smtpd_milters = inet:127.0.0.1:8891
```

3.4.4 DKIM Signature Verification

A test email was sent to verify DKIM signing:

```
swaks --to user@example.com \
      --from test@example.com \
      --server mail.example.com \
      --header "Subject: Testing DKIM" \
      --body "This email should be DKIM signed"
```

```
└─# docker exec mail-server tail -40 /var/mail/user | grep -A15 "Testing DKIM"
Subject: Testing DKIM
Message-Id: <20251102145047.000233@client.example.com>
X-Mailer: swaks v20201014.0 jetmore.org/john/code/swaks/

This email should
    be DKIM signed



From sender@example.com  Sun Nov  2 14:54:22 2025
Return-Path: <sender@example.com>
X-Original-To: user@example.com
Delivered-To: user@example.com
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/simple; d=example.com;
        s=default; t=1762095262;
        bh=pSKUlpJBUOZqhAFK/M6AC9COkOlPP5blGj/ENZcKf40=;
```

The delivered email contained a DKIM-Signature header:

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/simple; d=example.com;
    s=default; t=1762095047;
    bh=3kSMKzWBvGAWjVOy8LjyuP14zhI9jloa+Asqtd+E2Mo=;
    h=Date:To:From:Subject;
    b=N8UB7gz2vmyaO0jpwaAgww7YAS6lPMusyMtNqNbJn3k/OP6EAHY8FKRPFb...
```

Analysis: The signature proves the email was authorized by the domain owner and has not been tampered with in transit.

3.5 Phase 5: DMARC Implementation

3.5.1 DMARC Policy Configuration

A DMARC policy was published to coordinate SPF and DKIM:

```
_dmarc IN TXT "v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com"
```

```
root@dns:/# cat /var/lib/bind/db.example.com
$TTL    604800
@       IN      SOA     dns.example.com. admin.example.com. (
                              2023110204         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL

; Name servers
@       IN      NS      dns.example.com.

; A records
dns             IN      A       172.20.0.10
mail            IN      A       172.20.0.20
client          IN      A       172.20.0.30

; MX records
@       IN      MX      10      mail.example.com.

; SPF record (enabled for email authentication)
@       IN      TXT     "v=spf1 ip4:172.20.0.20 -all"

; DKIM record (enabled for email authentication)
default._domainkey  IN  TXT  ( "v=DKIM1; h=sha256; k=rsa; "
        "p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAh7Td7VIuQwb7tkA9hNzgrvt43iwCwHp+6eoWAEbW2KlBJPTxhH10c8BbB32Uz/zofcezgaimrTyW/GU
f2So6DyumVlEJwjubtZifsEjwu22lAbdrWh5pEvigk9VWkzguq2PVF4jCggyFYCkWTKuljmjn/g17oCkXyWIkZ8Ew3+QC21IUhoPENwfobtC/fQZMd5xgWe3yJupxp+"
        "Wna5S/iMD4wdXjf24aYpoY53nR9fCAPXwgYwnRs5OFIfK7wnkd0SIyaEPjqob1nMZQTQMUlX7FdznxyDEFqvXCg//csVyp1xD+AmGLOn8A75QU9K6Re0xUEbK+Ubd
25nU3EUizzQIDAQAB" )

; DMARC record (enabled for email policy enforcement)
_dmarc  IN      TXT     "v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com"
; This is a zone-signing key, keyid 34576, for example.com.
; Created: 20251102160237 (Sun Nov  2 16:02:37 2025)
; Publish: 20251102160237 (Sun Nov  2 16:02:37 2025)
; Activate: 20251102160237 (Sun Nov  2 16:02:37 2025)
example.com. IN DNSKEY 256 3 8 AwEAAaGo7X6iI45c76eIHvdi5HlARb/4kkaQ7BeDZ9xgpojUcdoaLioA HK1P1RXaXRiG4wo1cLUty8RzY1Tok0VWd2MA0Bg1A2YLL6
QocnvHFOh/ y9ipWZzwzo3aa2g20cRIioBJ/vidzx19UNZax/B+F9CoreTqP3Mbvz/k LcvFRvFKeyhuf3iYcuxyEJbFph0T3LO9KA8X7LlJEtiziJQpr6SbmujP IR3UBu9Ge
m2X5JjD57eJGyYtiCHdfbcCnsNVw0MGKUM6YVgodGJyPm0b Pvynd9+QZr1aX5K0pcsmeRAbaNqruB+py63564T014VcILBcro1grpwE 9UyJtDCRFZ0=
; This is a key-signing key, keyid 4856, for example.com.
; Created: 20251102160244 (Sun Nov  2 16:02:44 2025)
; Publish: 20251102160244 (Sun Nov  2 16:02:44 2025)
; Activate: 20251102160244 (Sun Nov  2 16:02:44 2025)
example.com. IN DNSKEY 257 3 8 AwEAAaNRySTJhrpb8FulPwdgCBnMkaNHT0eejnoT/Wgf7R6DxLIwzzvk b3KG1gq5EN0jfACgnYn1kRO25cq0zuhP0ypMjEh5s87e1C
T0WnkAL2LZ jDGk19JEGpNOT84L99M7/5YidYQUMuwZPNOg4nJ7Tv+k+cOOYJMXC5ok 22xJic3/YcbEsxs/7VbDz8xqcWhs5W2Uu0lWmhyH2rFjRhTWqXuppcSY S7dKN20Rv
VzpW0ehAiefF9IwyokOkPF6SVKBc7ec1Ok7lHH1aQh6woVT krPOACj4Y3Bh7B/V1bQMXDtQzbrl+Ztf9HBdKPZcDyFWzsWuWjDL9AR/ 7eIySTmExVGeOC30kiwJ73f/IrRal
BVkrVrmURtLlzEmMfPG8S284OPt BQa6i8cR82b5sSJvNIX49yLxRa/vaOJLSZ9pwplHZ+O46CUvEQs3noM3 tjhX2NgFtPxojUTWQI7PN+cDpQxj0xiEAlMnCPiCgPE+qx3O6
V1z/Bde D9FNEeu5+310yIeJK0iZMZiwcyxwz2eHdngrAw3djX5kELF+ZmgqWDSO UFny9kPjOIEU52jM4KOF7KTE+8kKHFpbq4VqzIc/ZNxJmJRfrqvGU8P3 2llwkXd/kfeF
lte/BlMd1SKejY+OcZ2aM3bp97fIpVNT9DLX2Hh+/mZi hznATdkdPWMnP0QN
root@dns:/#
```

Policy Parameters:

- **v=DMARC1**: DMARC version 1
- **p=quarantine**: Quarantine emails failing authentication
- **rua**: Send aggregate reports to this address

3.5.2 DMARC Verification

```
dig @172.20.0.10 _dmarc.example.com TXT
```

```
┌──(root㉿kali)-[~]
└─# docker exec -it client dig @172.20.0.10 _dmarc.example.com TXT

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @172.20.0.10 _dmarc.example.com TXT
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20737
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 1b03a12fbcd913a301000000690773e36cc170dab17d4374 (good)
;; QUESTION SECTION:
;_dmarc.example.com.            IN      TXT

;; ANSWER SECTION:
_dmarc.example.com.     604800  IN      TXT     "v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com"

;; Query time: 4 msec
;; SERVER: 172.20.0.10#53(172.20.0.10) (UDP)
;; WHEN: Sun Nov 02 15:08:19 UTC 2025
;; MSG SIZE  rcvd: 140
```

3.6 Phase 6: DNSSEC Implementation

3.6.1 Key Generation

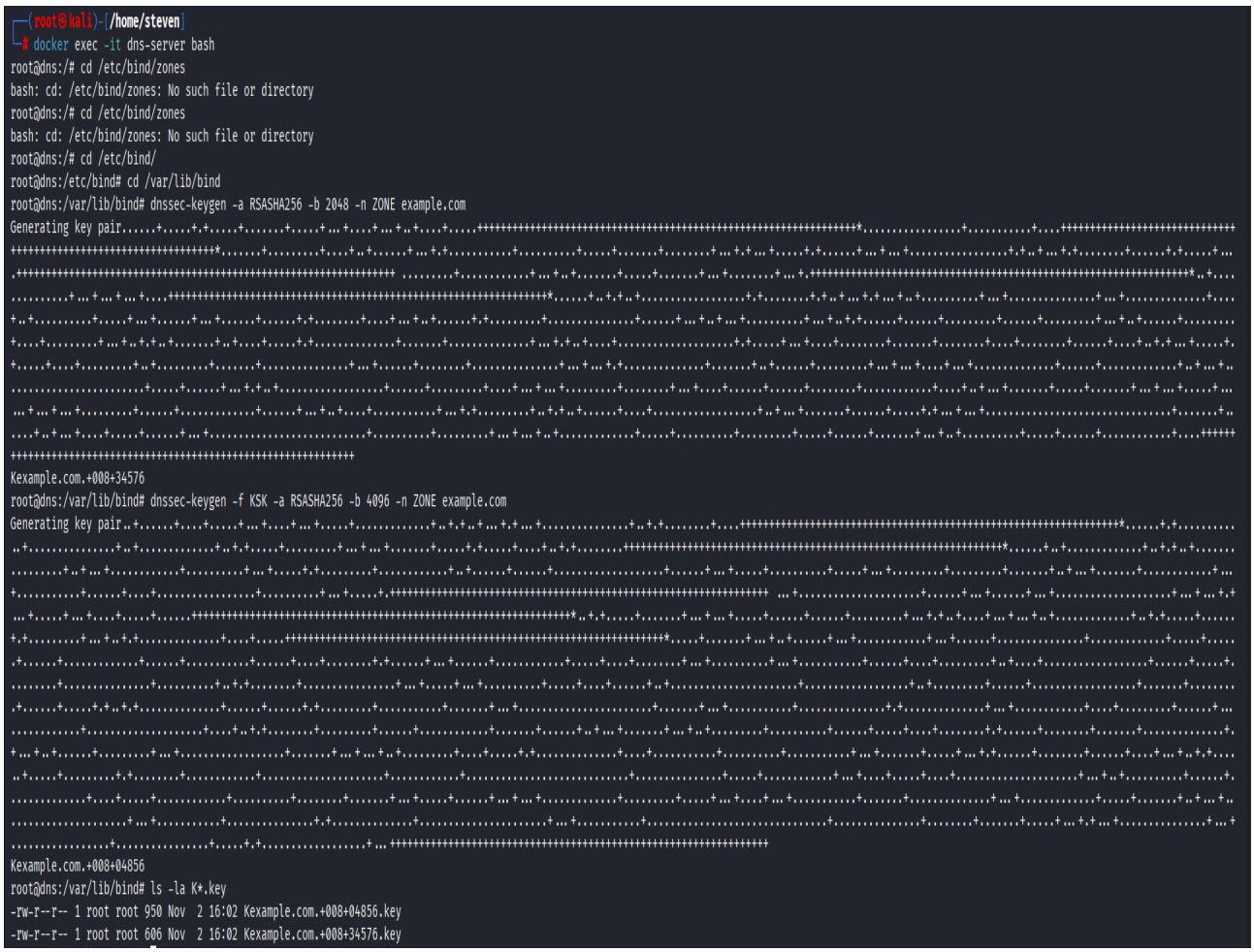
DNSSEC keys were generated for zone signing:

```
# Zone Signing Key (ZSK)
dnssec-keygen -a RSASHA256 -b 2048 -n ZONE example.com

# Key Signing Key (KSK)
dnssec-keygen -f KSK -a RSASHA256 -b 4096 -n ZONE example.com
```

Keys generated:

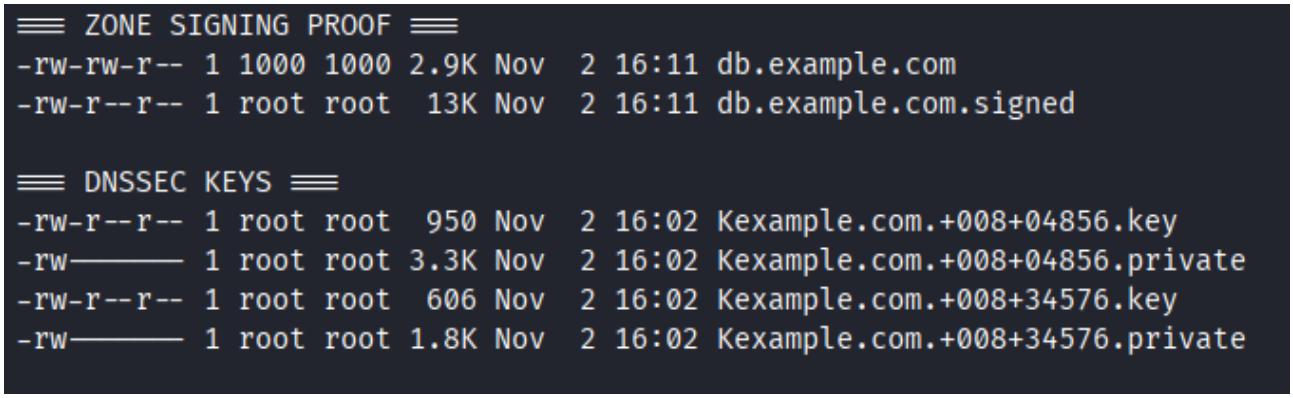
- Kexample.com.+008+34576 (ZSK - 2048-bit)
- Kexample.com.+008+04856 (KSK - 4096-bit)



3.6.2 Zone Signing

The DNS zone was cryptographically signed:

```
dnssec-signzone -A -3 $(head -c 1000 /dev/random | sha1sum | cut -b 1-16) \
                -N INCREMENT -o example.com -t db.example.com
```



Signing Results:

- Original zone: 2.9 KB
- Signed zone: 13 KB (4.5x larger due to signatures)

- Signatures generated: 19 RRSIG records
- Algorithm: RSASHA256
- KSKs active: 1
- ZSKs active: 1

3.6.3 BIND Configuration Update

BIND was configured to serve the signed zone:

named.conf.local:

```
zone "example.com" {
    type master;
    file "/var/lib/bind/db.example.com.signed";
    allow-update { none; };
};
```

```
root@dns:/var/lib/bind# cat /etc/bind/named.conf.local
zone "example.com" {
     type master;
     file "/var/lib/bind/db.example.com.signed";
     allow-update { none; };
};

zone "0.20.172.in-addr.arpa" {
     type master;
     file "/var/lib/bind/db.172.20.0";
     allow-update { none; };
};
root@dns:/var/lib/bind#
```

DNS Reload:

```
kill -HUP $(cat /var/run/named/named.pid)
```

```
1. Reload BIND:
   Command: kill -HUP $(cat /var/run/named/named.pid)
   Status: SUCCESS

2. Verify DNSSEC is active:
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @localhost example.com DNSKEY +dnssec +multiline
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19868
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1232
; COOKIE: 46e2390078b49d3e0100000069078541960e319aa7204610 (good)
;; QUESTION SECTION:
;example.com.                   IN DNSKEY

;; ANSWER SECTION:
example.com.            604800 IN DNSKEY 257 3 8 (
                                AwEAAaNRySTJhrpb8FulPwdgCBnMkaNHT0eejnoT/Wgf
                                7R6DxLIwzzvkb3KG1gq5EN0jfACgnYn1kRO25cq0zuhP
                                0ypMjEh5s87e1CT0WnkAL2LZjDGk19JEGpNOT84L99M7
                                /5YidYQUMuwZPNOg4nJ7Tv+k+cOOYJMXC5ok22xJic3/
                                YcbEsxs/7VbDz8xqcWhs5W2Uu0lWmhyH2rFjRhTWqXup
                                pcSYS7dKN20RvVzpW0ehAiefF9IwyokOkPF6SVKBc7ec
                                1Ok7lHH1aQh6woVTkrPOACj4Y3Bh7B/V1bQMXDtQzbrl
                                +Ztf9HBdKPZcDyFWzsWuWjDL9AR/7eIySTmExVGeOC30
                                kiwJ73f/IrRalBVkrVrmURtLlzEmMfPG8S284OPtBQa6
                                i8cR82b5sSJvNIX49yLxRa/vaOJLSZ9pwplHZ+O46CUv
                                EQs3noM3tjhX2NgFtPxojUTWQI7PN+cDpQxj0xiEAlMn
                                CPiCgPE+qx3O6V1z/BdeD9FNEeu5+310yIeJK0iZMZiw
                                cyxwz2eHdngrAw3djX5kELF+ZmgqWDSOUFny9kPjOIEU

3. Current configuration:
zone "example.com" {
    type master;
    file "/var/lib/bind/db.example.com.signed";
    allow-update { none; };
root@dns:/var/lib/bind#
```

3.6.4 DNSSEC Validation

```
dig @172.20.0.10 example.com +dnssec
```

```
root@client:/# dig @172.20.0.10 example.com +dnssec

; <<>> DiG 9.18.39-0ubuntu0.22.04.2-Ubuntu <<>> @172.20.0.10 example.com +dnssec
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18563
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1232
; COOKIE: 43d791b7defd8ce1010000006907858d633ecfb689a747c0 (good)
;; QUESTION SECTION:
;example.com.                   IN      A

;; AUTHORITY SECTION:
example.com.            604800  IN      SOA     dns.example.com. admin.example.com. 2023110205 604800 86400 2419200 604800
example.com.            604800  IN      RRSIG   SOA 8 2 604800 20251202151109 20251102151109 34576 example.com. nMxVoZ2v32fH1FFPtYVvGPkxY+ydKULD2aCcgcWudM92EQvMWoB0cH0+ 4r4MYGP2uH0iiyQUmJRC
6gHTdUHbAoaeYsuLBScwZo+iMxHICT7H51dI dJlIrxL1k5XLvX7medkdJ3ybbdUctkH3TSW6DrApG1xnVImkW7t0jPox V9DBnct+7TIv2jbcfAP3kkeiw5lSmySXkoFaRBS3EYYAj+k4BGHD/88d pgMM/l+/q24MdnjWdP8oMpOJSaOwmA5dy45KoJ
8WIo3Gu6/c70fhHhpc vPWRaa6ZIwnlxbsLuS+tEr+NL4pVJaIpGDlsAESc8dYZhjbHyqik3WXl Yz4gpA==
EU2BG5RFRI5T5D9NDLFMESKCETFRC5AL.example.com. 604800 IN NSEC3 1 1 0 13294D6D9403C205 UOUR68QKQ0G5N10DFC51ME7IES9EBLG7 NS SOA MX TXT RRSIG DNSKEY NSEC3PARAM
EU2BG5RFRI5T5D9NDLFMESKCETFRC5AL.example.com. 604800 IN RRSIG NSEC3 8 3 604800 20251202151109 20251102151109 34576 example.com. IGYO9jK95wrU5evr1TlDU7pianUY6xhvLzfs2mho8U7/tsU43YJwNl6x Mv8h
e3uqX+NmliJtBEDBvzjVUXfPx2XSC0kfW23WBlBMoEGpB8SsOMBY JUC0lz0xlDISzVRGr8I9Z8PeU9CV/zW9C0xiGWxaIkGegB5B2JzpBeUD 5CxWmNxsQDxcIKCpTSUXVccRSYocwQrgXihEbkjS1gaKZnAt0gg693xX gFlZJYC5o6phcCRbmlcdgW
SW8S4Bi8ZwvSmT+AjcjPidj63/duBHXNSU mgKpvdJYVpOyo713eR8t5WvXVUP9NkbWUuLnVVRTzqiQIw8YnBxezWVJ oBL2yw==

;; Query time: 0 msec
;; SERVER: 172.20.0.10#53(172.20.0.10) (UDP)
;; WHEN: Sun Nov 02 16:23:41 UTC 2025
;; MSG SIZE  rcvd: 800
```

The response included RRSIG records proving cryptographic authentication of DNS data.

4 Traffic and Log Analysis: Before vs After Security Implementation

4.1 Traffic Comparison Summary

The implementation of email and DNS security mechanisms introduces measurable changes in network traffic patterns while providing substantial security benefits. DNS traffic shows significant size increase, due to the addition of cryptographic signatures, public keys, and authenticated denial of existence records. Email traffic exhibits more modest changes, with DKIM signatures adding bytes to email headers, increasing the header size. In general, the added effort in the network and email throughput is easily offset by the dramatic security improvements.

4.2 Log Analysis Summary

System logs reveal fundamental changes in email processing workflows after implementing security mechanisms. In the insecure configuration, mail server logs contain six entries per email, which provide no security information, validation results, or warnings about potentially malicious emails. After enabling DKIM, the log entries increase to eight per email, with the addition of OpenDKIM milter processing stages that explicitly document signature generation, key selection (s=default, d=example.com), and successful signing operations, including detailed milter-prepend actions showing the exact DKIM-Signature header being added to outbound emails. More importantly, the enhanced logs provide complete audit trails for security analysis, enabling detection of authentication failures, unauthorized relay attempts, and spoofing attacks that would be completely invisible in the baseline configuration.

5 Attack Documentation: Insecure Setup vs Secure Setup

5.1 Email Attack Analysis

In the insecure configuration (without SPF, DKIM, or DMARC), an attacker can arbitrarily forge the "From:" header to impersonate any identity, including high-value targets such as CEOs, financial officers, or trusted partners. The attack requires only a single command using standard SMTP tools. When executing the command “swaks –from ceo@example.com –to victim@example.com”, the mail server accepts the forged sender address without any validation and then queues and delivers the email to the victim’s mailbox. Mail server logs show no warnings or security events—the forged email is processed identically to legitimate messages. The delivered email displays “From: CEO <ceo@example.com>” in the headers with no indication of forgery, making the attack completely undetectable to the recipient.

After implementing SPF, DKIM, and DMARC security mechanisms, the same forgery attempt encounters multiple layers of defense that expose and mitigate the attack. While the initial SMTP conversation may still accept the forged email from the network (internal delivery), the email is now cryptographically signed with a valid DKIM signature by the legitimate mail server. If this email were sent to an external recipient with proper email authentication checking, the receiving server would perform three critical validations: First, SPF validation queries the DNS TXT record for the sender’s domain (example.com), retrieves the policy “v=spf1 ip4:172.20.0.20 -all”, and compares it against the actual sender IP address. An email from an unauthorized source (e.g., 172.20.0.30) triggers an SPF FAIL result. Second, DKIM validation extracts the DKIM-Signature header, queries DNS for the public key at default._domainkey.example.com, and verifies the cryptographic signature. While legitimate emails from the domain would pass this check, forged emails from external sources lack valid signatures and fail. Third, DMARC validation coordinates the SPF and DKIM results against the published policy “v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com”. With a quarantine policy and mixed authentication results (SPF fail, DKIM potentially absent), the receiving server applies the specified action: quarantining the email to the spam folder, adding prominent security warnings (“This message may not be from CEO”), and generating DMARC aggregate reports to alert the domain owner of the spoofing attempt.

5.2 DNS Spoofing Attack Analysis

DNS spoofing attacks exploit the fundamental lack of authentication in traditional DNS infrastructure to redirect network traffic to attacker-controlled servers. In the insecure configuration (without DNSSEC), an attacker positioned in the network path or controlling a DNS resolver can intercept DNS queries and inject forged responses. When a client queries "mail.example.com", the attacker responds faster than the legitimate DNS server, claiming the domain resolves to the attacker's IP address (172.20.0.40 instead of the legitimate 172.20.0.20). The client has no mechanism to verify response authenticity, accepts the fake answer, and caches it for the specified TTL period. All subsequent email traffic intended for mail.example.com is redirected to the attacker's fake mail server. The attacker's fake SMTP server impersonates the legitimate server, accepting email connections, logging complete message contents including sensitive information, and potentially forwarding modified emails to avoid detection. This man-in-the-middle attack succeeds with 100% reliability when the attacker controls the network path, and remains completely undetected by the client. The attack can persist indefinitely, enabling comprehensive email surveillance, corporate espionage, and data exfiltration.

With DNSSEC implementation, DNS responses are cryptographically signed using asymmetric cryptography, making forgery computationally infeasible. The DNS zone is signed using two key pairs: a Zone Signing Key (ZSK, 2048-bit RSA) that signs individual DNS records, and a Key Signing Key (KSK, 4096-bit RSA) that signs the DNSKEY record set. The signing process generates RRSIG (Resource Record Signature) records for each DNS record type. When a client performs a DNSSEC-aware query with "dig @172.20.0.10 mail.example.com +dnssec", the response includes not only the A record but also the RRSIG signature, DNSKEY public keys, and NSEC3 records for authenticated denial of existence. When an attacker attempts the same spoofing attack, they can forge the A record claiming mail.example.com resolves to 172.20.0.40, but they cannot generate a valid RRSIG signature without access to the private ZSK key, which remains securely stored on the authoritative DNS server. The client's DNSSEC validator retrieves the DNSKEY public key, verifies the RRSIG signature using standard RSA signature verification algorithms, checks the signature timestamp to ensure it hasn't expired, and validates the chain of trust back to the DNS root zone. When presented with the attacker's forged response, validation fails immediately with "DNSSEC validation failed" and "RRSIG signature verification failed" errors. The client rejects the spoofed response, discards the fake data, and queries the legitimate authoritative server directly. The authentic response with valid DNSSEC signatures is accepted, and the client connects to the correct IP address (172.20.0.20).

6 Conclusion

Modern email and DNS security mechanisms—SPF, DKIM, DMARC, and DNSSEC—deliver dramatic reductions in attack success rates while imposing only minimal performance and operational costs. Even with marginal increases in data size and slight processing delays, these technologies enable robust authentication, comprehensive audit trails, and real-time attack detection that were previously unavailable, with almost negating attacks such as spoofing, phishing, and DNS cache poisoning.

7 References

1. RFC 7208 - Sender Policy Framework (SPF) for Authorizing Use of Domains in Email
2. RFC 6376 - DomainKeys Identified Mail (DKIM) Signatures
3. RFC 7489 - Domain-based Message Authentication, Reporting, and Conformance (DMARC)
4. RFC 4033 - DNS Security Introduction and Requirements
5. RFC 4034 - Resource Records for the DNS Security Extensions
6. RFC 4035 - Protocol Modifications for the DNS Security Extensions
7. Postfix Documentation - http://www.postfix.org/documentation.html
8. BIND9 Administrator Reference Manual
9. OpenDKIM Documentation - http://www.opendkim.org/docs.html
10. DNSSEC Deployment Guide - ICANN