SUNIMOD FIELD NOTES

Your Email Domain Can Be Forged: How Spoofing Turns Into Business Email Compromise.

Learn how email domain spoofing abuses SPF, DKIM, and DMARC gaps, what it can cost a business, how to audit it safely, and how Sunimod can help remediate the risk.

BY Christopher Jacobson

PUBLISHED July 15, 2026

TOPIC CISSP, Domain 1

https://sunimod.com/your-email-domain-can-be-forged-how-spoofing-turns-into-business-email-compromise/

In this guide

The vulnerability in plain language

The visible sender address in an email is not the same thing as proof of who sent it. Traditional email transport allows several identities to exist in one message. The address a person sees in the `From:` line can differ from the address used for delivery, and a cryptographic signature can belong to yet another domain.

This creates a dangerous opening when a company has no effective domain-authentication policy, has a monitoring-only policy, or has not aligned all of its legitimate sending services. An attacker can place the company's domain in the visible `From:` field and rely on the recipient's mail system, the recipient's habits, or both to let the message through.

> **This is usually not a software bug or a dramatic server intrusion.** It is an abuse of email's trust model combined with incomplete DNS configuration, unmanaged third-party senders, and business processes that treat an email as sufficient proof for a sensitive decision.

The weakness becomes especially valuable to criminals when the organization regularly sends invoices, receives wire instructions, changes payroll data, resets passwords, delivers account alerts, or communicates with customers through automated website and application email.

The solvable business opportunity: an Email Identity and BEC Risk Review

This problem can be turned into a focused, valuable service rather than an open-ended security project. A business such as Sunimod can package the work as an **Email Identity and Business Email Compromise Risk Review** that connects domain configuration to the real systems and workflows that send messages.

Discover

Inventory domains, subdomains, mailboxes, website forms, ecommerce systems, CRMs, ticketing tools, accounting platforms, marketing services, scanners, and every vendor that sends on the company's behalf.

Align

Correct SPF authorization, enable provider-specific DKIM signing, align visible sender domains, separate risky bulk mail, and establish DMARC reporting without interrupting legitimate delivery.

Enforce and monitor

Move safely from observation to quarantine and rejection, harden inbound impersonation controls, document payment-change procedures, and review reports as vendors and systems change.

The customer is not buying three DNS records. The customer is buying a verified map of who is allowed to speak for the company, a controlled rollout that avoids breaking real mail, and a process that keeps a convincing message from becoming an unauthorized payment.

How email identity actually works

1. The visible author: `From:`

This is the identity most people see in Outlook, Gmail, Apple Mail, and mobile notification previews. It is the identity attackers want to borrow because it carries the company name, executive name, or vendor relationship that makes a request believable.

2. The envelope sender: SMTP `MAIL FROM` and `Return-Path`

Mail servers use an envelope address for delivery status and bounce handling. SPF evaluates the domain tied to this SMTP identity. The envelope can legitimately differ from the visible author when a cloud platform or transactional service sends on behalf of a customer.

3. The DKIM signing domain: the `d=` value

DKIM adds a cryptographic signature to selected message headers and the body. The receiving system retrieves a public key from DNS and verifies that the signed content has not been changed. The signing domain is declared in the DKIM `d=` tag.

4. DMARC alignment: binding authentication to what the person sees

SPF can pass for one domain and DKIM can pass for another without proving that either domain matches the visible author. DMARC closes that gap by asking whether at least one authenticated domain aligns with the visible `From:` domain.

Plain text · DMARC alignment logic

```
DMARC passes when either path succeeds:

(SPF passes AND the SPF-authenticated domain aligns with the visible From domain)
OR
(DKIM passes AND the DKIM d= signing domain aligns with the visible From domain)
```

Relaxed alignment generally accepts subdomains under the same organizational domain. Strict alignment requires an exact domain match. A DMARC pass establishes that the domain's use was authorized; it does *not* prove that the content is truthful or harmless. A compromised mailbox can send malicious mail that authenticates correctly.

How an email spoofing attack unfolds

1

Reconnaissance creates the story

The attacker studies the company's website, staff directory, social media, job roles, public contracts, vendor names, invoice language, and timing. A polished pretext often matters more than technical sophistication.

2

The attacker chooses an identity technique

Exact-domain spoofing places the real company domain in the visible sender field without authorization. **Lookalike-domain impersonation** uses a newly registered spelling variation. **Mailbox compromise** uses a real account after credential theft or token theft. DMARC is strongest against the first technique, can help reputation analysis around the second, and does not by itself stop the third.

3

A legitimate-looking message is assembled

The attacker may use their own properly configured sending domain for the technical envelope and signature while placing the victim company's domain in the visible `From:` field. SPF and DKIM can both pass for the attacker's infrastructure while DMARC correctly fails because neither authenticated domain aligns with the visible author.

Plain text · illustrative message headers

```
From: "Accounts Payable" <billing@trusted-business.example>
Return-Path: <bounce@mailer.attacker.example>
DKIM-Signature: v=1; a=rsa-sha256; d=mailer.attacker.example;
 s=mail2026; h=from:to:subject:date; bh=...; b=...
Authentication-Results: mx.recipient.example;
 spf=pass smtp.mailfrom=mailer.attacker.example;
 dkim=pass header.d=mailer.attacker.example;
 dmarc=fail header.from=trusted-business.example
Subject: Updated banking details for invoice 48317
```

Read the example from bottom to top:

- The recipient sees a billing address at `trusted-business.example`.
- SPF passes only for `mailer.attacker.example`, the envelope domain.
- DKIM passes only for `mailer.attacker.example`, the signing domain.
- DMARC fails because the authenticated domains do not align with `trusted-business.example`.

When the protected domain publishes an enforcement policy, it requests that participating receivers quarantine or reject this failure. When no DMARC record exists—or the policy remains p=none—the domain owner has not requested that enforcement. The receiving provider may still block the message using other signals, but the company has surrendered one of its clearest controls over unauthorized use of its own name.

Do not confuse exact-domain spoofing with a lookalike domain. A criminal who registers a visually similar domain can publish valid SPF, DKIM, and DMARC for that domain. The message may authenticate perfectly while still deceiving the reader. Inbound impersonation protection, domain monitoring, user interface warnings, and verified business processes remain necessary.

4

Urgency suppresses verification

The message asks for a bank-account change, confidential document, gift-card purchase, payroll update, password reset, or login. It may claim that an executive is in a meeting, a vendor is about to stop shipment, or a payment deadline will be missed.

5

The business process becomes the final vulnerability

Even perfect filtering cannot guarantee that every deceptive message is blocked. The attacker wins when one person can change payment details or release funds based only on an email. Technical controls and approval controls must reinforce each other.

What SPF, DKIM, and DMARC do—and where each one stops

SPF: which systems may send for the envelope domain

SPF is a DNS policy that lists permitted sending sources for the SMTP envelope domain. It is useful, but it does not compare that domain with the visible `From:` domain. That is why an attacker can authorize their own infrastructure, pass SPF for their own domain, and still display a different company's address to the user.

SPF also has an evaluation limit of ten DNS-querying mechanisms and modifiers. Deep chains of `include:` statements can exceed that limit and produce a permanent error. A second SPF record does not increase capacity; multiple SPF policies can make the domain invalid.

DKIM: a verifiable signature attached by a responsible domain

DKIM signs selected content using a private key. The public key is published under a selector such as `selector1._domainkey.example.com`. It helps recipients verify integrity and associate a responsible domain with the message. Forwarding often breaks SPF because the sending IP changes, while DKIM may survive if the signed content is not modified.

DMARC: policy, alignment, and reporting

DMARC publishes a TXT policy under `_dmarc`. It evaluates alignment with the visible author domain, lets the domain owner request `none`, `quarantine`, or `reject` treatment for failures, and can request aggregate reports that reveal legitimate senders and apparent abuse.

Policy	What the domain owner requests	Appropriate use	Main limitation
`p=none`	Collect and evaluate results without requesting quarantine or rejection.	Initial discovery and controlled troubleshooting.	It is visibility, not enforcement.
`p=quarantine`	Treat failures as suspicious, commonly through spam or quarantine handling.	Intermediate containment after legitimate senders are mostly aligned.	Bad mail may still reach a user depending on receiver behavior.
`p=reject`	Reject messages that fail DMARC.	Target state after sender inventory, alignment, and monitoring.	Receivers retain discretion, and authentication does not stop lookalikes or compromised accounts.

A safer target state: one maintained SPF policy, DKIM enabled for every legitimate sending service, DMARC at enforcement, dedicated subdomains for high-volume or externally operated senders, parked domains set to send no mail, inbound impersonation controls, strong account protection, and payment changes verified outside the email thread.

Safe Ubuntu checks you can run without sending an email

The following commands perform read-only DNS lookups. They do not attempt to spoof a message, log in to a provider, or change a record. Only assess domains you own or are authorized to review.

Bash · read-only DNS checks on Ubuntu

```
sudo apt update
sudo apt install -y dnsutils

read -r -p "Enter the domain you are authorized to audit: " DOMAIN
DOMAIN="$(printf '%s' "$DOMAIN" \
  | sed -E 's#^[A-Za-z][A-Za-z0-9+.-]*://##; s#/.*$##; s/\.$//' \
  | tr '[:upper:]' '[:lower:]')"

if [[ ! "$DOMAIN" =~ ^([a-z0-9]([a-z0-9-]{0,61}[a-z0-9])?\.)+[a-z]{2,63}$ ]]; then
  printf 'Invalid domain name: %s\n' "$DOMAIN" >&2
  exit 2
fi

printf '\nSPF records for %s\n' "$DOMAIN"
dig +short TXT "$DOMAIN" | grep -i 'v=spf1' || printf 'No SPF record found.\n'

printf '\nDMARC record for %s\n' "$DOMAIN"
dig +short TXT "_dmarc.$DOMAIN" | grep -i 'v=DMARC1' || printf 'No DMARC record found.\n'

read -r -p "Enter a DKIM selector, or press Enter to skip: " SELECTOR
if [[ -n "$SELECTOR" ]]; then
  if [[ ! "$SELECTOR" =~ ^[A-Za-z0-9]([A-Za-z0-9-]{0,61}[A-Za-z0-9])?$ ]]; then
    printf 'Invalid DKIM selector: %s\n' "$SELECTOR" >&2
    exit 2
  fi

  printf '\nDKIM record for selector %s\n' "$SELECTOR"
  dig +short TXT "${SELECTOR}._domainkey.${DOMAIN}"
  dig +short CNAME "${SELECTOR}._domainkey.${DOMAIN}"
fi

printf '\nMX records for %s\n' "$DOMAIN"
dig +short MX "$DOMAIN"
```

How to interpret the first results

- **No SPF record:** the domain is not publishing a sender authorization policy for its envelope identity.
- **More than one SPF record:** consolidate them into one policy rather than publishing multiple `v=spf1` records.
- **No DMARC record:** the domain is not publishing an aligned author-domain policy at the expected location.
- `p=none`**:** reports may be requested, but failures are not under a domain-owner-requested quarantine or rejection policy.
- **No DKIM result:** the selector may be wrong, the provider may use a CNAME, or DKIM may not be enabled. Obtain the selector from a real message header or the provider's administration portal.

Illustrative DNS records

These examples use reserved documentation domains. They demonstrate structure, not a universal configuration. An active domain must be based on a complete sender inventory before enforcement.

DNS · illustrative DMARC monitoring record

```
; Illustrative active-domain monitoring record
_dmarc.example.com. 3600 IN TXT
"v=DMARC1; p=none; rua=mailto:dmarc-reports@example.com; adkim=r; aspf=r"

; Illustrative active-domain enforcement record after all senders are aligned
_dmarc.example.com. 3600 IN TXT
"v=DMARC1; p=reject; sp=reject; rua=mailto:dmarc-reports@example.com; adkim=r; aspf=r"

; A parked domain that must never send email
example.net.        3600 IN TXT "v=spf1 -all"
_dmarc.example.net. 3600 IN TXT "v=DMARC1; p=reject; sp=reject"
```

An SPF record is specific to the services that actually send mail. The following record is correct only for an organization whose sole authorized sending platform is Microsoft 365:

DNS · illustrative SPF record

```
; Correct only when Microsoft 365 is the sole authorized sending service
example.com. 3600 IN TXT "v=spf1 include:spf.protection.outlook.com -all"
```

> A website contact form, ecommerce platform, CRM, support desk, accounting service, newsletter provider, copier, alerting platform, or custom application may send outside the primary mail provider. Publishing a strict record before those systems are identified can break legitimate delivery.

A complete read-only Python audit tool

This Ubuntu-compatible tool checks public MX, SPF, DMARC, and supplied DKIM selector records. It flags common configuration problems, shows a top-level estimate of SPF lookup-causing terms, and never sends email or modifies DNS.

Bash · run the read-only audit

```
sudo apt update
sudo apt install -y python3-venv

python3 -m venv .venv
source .venv/bin/activate
python -m pip install --upgrade pip dnspython

python email_domain_audit.py sunimod.com
python email_domain_audit.py sunimod.com \
  --selector selector1 \
  --selector selector2
```

Open the complete `email_domain_audit.py` source

Python · read-only email-domain audit

```
#!/usr/bin/env python3
"""Read-only SPF, DKIM, and DMARC DNS audit."""

from __future__ import annotations

import argparse
import re
import sys

try:
    import dns.exception
```

```
    import dns.resolver
except ImportError:
    print(
        "dnspython is required.\n"
        "Ubuntu setup:\n"
        "  python3 -m venv .venv\n"
        "  source .venv/bin/activate\n"
        "  python -m pip install --upgrade pip dnspython",
        file=sys.stderr,
    )
    raise SystemExit(2)

LABEL_RE = re.compile(r"^[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?$", re.I)
EXIT_WARNING = False


def finding(level: str, title: str, detail: str) -> None:
    global EXIT_WARNING
    if level in {"HIGH", "WARN"}:
        EXIT_WARNING = True
    print(f"[{level:<4}] {title}\n       {detail}")


def normalize_domain(value: str) -> str:
    value = re.sub(r"^[a-z][a-z0-9+.-]*://", "", value.strip().lower())
    value = value.split("/", 1)[0].split(":", 1)[0].rstrip(".")
    try:
        value = value.encode("idna").decode("ascii")
    except UnicodeError as exc:
        raise ValueError(f"Invalid internationalized domain: {exc}") from exc
    labels = value.split(".")
    if len(labels) < 2 or len(value) > 253:
        raise ValueError("Enter a registrable DNS domain.")
    if any(not LABEL_RE.fullmatch(label) for label in labels):
        raise ValueError("The domain contains an invalid DNS label.")
    return value


def txt_records(resolver: dns.resolver.Resolver, name: str) -> list[str]:
    try:
        answers = resolver.resolve(name, "TXT")
    except (
        dns.resolver.NXDOMAIN,
        dns.resolver.NoAnswer,
        dns.resolver.NoNameservers,
        dns.exception.Timeout,
    ):
        return []

    records: list[str] = []
    for answer in answers:
        if hasattr(answer, "strings"):
            records.append(b"".join(answer.strings).decode("utf-8", "replace"))
        else:
            text = answer.to_text()
            chunks = re.findall(r'"([^"]*)"', text)
            records.append("".join(chunks) if chunks else text.strip('"'))
    return records


def cname_target(resolver: dns.resolver.Resolver, name: str) -> str | None:
    try:
        answers = resolver.resolve(name, "CNAME")
    except (
        dns.resolver.NXDOMAIN,
        dns.resolver.NoAnswer,
        dns.resolver.NoNameservers,
        dns.exception.Timeout,
    ):
        return None
    return str(answers[0].target).rstrip(".") if answers else None
```

```
def tag_list(record: str) -> dict[str, str]:
    tags: dict[str, str] = {}
    for item in record.split(";"):
        if "=" in item:
            key, value = item.split("=", 1)
            tags[key.strip().lower()] = value.strip()
    return tags


def spf_lookup_terms(record: str) -> list[str]:
    terms: list[str] = []
    for raw in record.split()[1:]:
        token = raw.lstrip("+-~?").lower()
        if (
            token == "a"
            or token.startswith(("a:", "a/"))
            or token == "mx"
            or token.startswith(("mx:", "mx/"))
            or token == "ptr"
            or token.startswith(("ptr:", "include:", "exists:", "redirect="))
        ):
            terms.append(raw)
    return terms


def inspect_spf(domain: str, resolver: dns.resolver.Resolver) -> None:
    print("\nSPF\n---")
    records = [
        item
        for item in txt_records(resolver, domain)
        if item.lower().startswith("v=spf1")
    ]
    if not records:
        finding("HIGH", "No SPF record", f"No v=spf1 record exists for {domain}.")
        return
    if len(records) > 1:
        finding(
            "HIGH",
            "Multiple SPF records",
            "Publish one SPF policy; multiple policies can produce PermError.",
        )

    for record in records:
        finding("INFO", "Published policy", record)
        lookups = spf_lookup_terms(record)
        finding(
            "INFO",
            "Top-level DNS lookup terms",
            f"{len(lookups)}: {', '.join(lookups) if lookups else 'none'}",
        )
        if len(lookups) > 10:
            finding(
                "HIGH",
                "SPF lookup limit exceeded",
                "More than 10 lookup-causing terms are visible before recursive "
                "include and redirect processing.",
            )
        elif len(lookups) >= 8:
            finding(
                "WARN",
                "SPF is close to the lookup limit",
                "Nested include records may push evaluation over the limit.",
            )

        tokens = {item.lower() for item in record.split()[1:]}
        if "all" in tokens or "+all" in tokens:
            finding("HIGH", "SPF authorizes all senders", "+all defeats the allow list.")
        elif "?all" in tokens:
            finding("WARN", "SPF ends neutral", "?all offers little protection.")
        elif "~all" in tokens:
            finding(
```

```
                "WARN",
                "SPF ends soft fail",
                "Use -all only after every legitimate sender is inventoried and tested.",
            )
        elif "-all" in tokens:
            finding("PASS", "SPF ends hard fail", "Unknown sources receive SPF fail.")
        else:
            finding("WARN", "No all mechanism", "The policy has no catch-all decision.")


def inspect_dmarc(domain: str, resolver: dns.resolver.Resolver) -> None:
    print("\nDMARC\n-----")
    name = f"_dmarc.{domain}"
    records = [
        item
        for item in txt_records(resolver, name)
        if item.lower().startswith("v=dmarc1")
    ]
    if not records:
        finding("HIGH", "No DMARC policy", f"No v=DMARC1 record exists at {name}.")
        return
    if len(records) > 1:
        finding(
            "HIGH",
            "Multiple DMARC policies",
            "Multiple policies at one DNS name are invalid.",
        )
        return

    record = records[0]
    tags = tag_list(record)
    finding("INFO", "Published policy", record)

    policy = tags.get("p", "").lower()
    if policy == "reject":
        finding("PASS", "Enforcement policy", "p=reject requests rejection.")
    elif policy == "quarantine":
        finding("WARN", "Partial enforcement", "p=quarantine requests suspicious handling.")
    elif policy == "none":
        finding("WARN", "Monitoring only", "p=none requests no enforcement.")
    else:
        finding("HIGH", "Invalid policy", "The p tag must be none, quarantine, or reject.")

    effective_subdomain = tags.get("sp", policy).lower()
    if effective_subdomain == "none":
        finding("WARN", "Subdomain policy", "Subdomains are effectively monitoring only.")
    else:
        finding(
            "INFO",
            "Subdomain policy",
            f"Effective policy: {effective_subdomain or 'invalid'}.",
        )

    if "rua" in tags:
        finding("INFO", "Aggregate reporting", tags["rua"])
    else:
        finding("WARN", "No rua destination", "Aggregate visibility is not requested.")

    finding(
        "INFO",
        "Alignment",
        f"DKIM={tags.get('adkim', 'r')} and SPF={tags.get('aspf', 'r')} "
        "(r=relaxed, s=strict).",
    )
    if "pct" in tags:
        finding(
            "WARN",
            "Legacy pct tag",
            "RFC 9989 marks pct historic; review rollout assumptions.",
        )
```

```
def inspect_dkim(
    domain: str,
    selectors: list[str],
    resolver: dns.resolver.Resolver,
) -> None:
    print("\nDKIM\n----")
    if not selectors:
        finding(
            "INFO",
            "Selector check skipped",
            "Supply --selector values from a DKIM-Signature s= tag or mail-provider
                portal.",
        )
        return

    for selector in selectors:
        selector = selector.strip()
        if not LABEL_RE.fullmatch(selector):
            finding("WARN", f"Invalid selector {selector!r}", "Use one DNS label.")
            continue

        name = f"{selector}._domainkey.{domain}"
        records = txt_records(resolver, name)
        target = cname_target(resolver, name)

        if target:
            finding("PASS", f"Selector {selector}", f"CNAME points to {target}.")
            continue
        if not records:
            finding("WARN", f"Selector {selector}", f"No TXT or CNAME record at {name}.")
            continue

        valid = False
        for record in records:
            tags = tag_list(record)
            if tags.get("p"):
                valid = True
                finding("PASS", f"Selector {selector}", "A DKIM public key is published.")
            else:
                finding("WARN", f"Selector {selector}", f"Record has no p value: {record}")
        if not valid:
            finding("WARN", f"Selector {selector}", "No usable public key was found.")


def show_mx(domain: str, resolver: dns.resolver.Resolver) -> None:
    print("\nMX\n--")
    try:
        answers = resolver.resolve(domain, "MX")
    except (
        dns.resolver.NXDOMAIN,
        dns.resolver.NoAnswer,
        dns.resolver.NoNameservers,
        dns.exception.Timeout,
    ):
        finding("WARN", "No MX records returned", "Confirm whether the domain receives
            mail.")
        return
    for answer in sorted(answers, key=lambda item: item.preference):
        finding(
            "INFO",
            "Mail exchanger",
            f"{answer.preference} {str(answer.exchange).rstrip('.')}",
        )


def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="Audit public SPF, DKIM, and DMARC DNS records without sending mail."
    )
    parser.add_argument("domain", help="Domain to inspect.")
    parser.add_argument(
        "--selector",
```

```
        action="append",
        default=[],
        help="DKIM selector; repeat this option to test more than one.",
    )
    parser.add_argument(
        "--timeout",
        type=float,
        default=4.0,
        help="DNS timeout in seconds (default: 4.0).",
    )
    return parser.parse_args()


def main() -> int:
    args = parse_args()
    try:
        domain = normalize_domain(args.domain)
    except ValueError as exc:
        print(f"Input error: {exc}", file=sys.stderr)
        return 2

    resolver = dns.resolver.Resolver()
    resolver.timeout = max(args.timeout, 1.0)
    resolver.lifetime = max(args.timeout, 1.0)

    print("Email Domain Authentication Audit")
    print("================================")
    print(f"Domain: {domain}")
    print("Mode: read-only public DNS")

    show_mx(domain, resolver)
    inspect_spf(domain, resolver)
    inspect_dmarc(domain, resolver)
    inspect_dkim(domain, args.selector, resolver)

    print(
        "\nScope note\n----------\n"
        "DNS alone cannot prove that every legitimate sender is aligned. "
        "Validate real message headers and DMARC aggregate reports before "
        "moving an active domain to enforcement."
    )
    return 1 if EXIT_WARNING else 0


if __name__ == "__main__":
    raise SystemExit(main())
```

Important limitation: DNS inspection cannot prove that every real sender is aligned. A production review must also inspect genuine message headers, DMARC aggregate reports, vendor documentation, application settings, and the organization's actual sending paths.

Potential repercussions for your business or infrastructure

Repercussion	What can happen	Why the impact spreads
Fraudulent wire, ACH, or invoice payment	An employee or customer sends funds to an account controlled by the criminal.	Recovery windows can be short, and responsibility may become disputed among the sender, recipient, vendors, banks, and insurers.
Payroll diversion	A fake employee request changes direct-deposit information.	The company may need to make the employee whole while investigating how the request was approved.
Credential theft and account takeover	A spoofed alert links to a fake login page or convinces support staff to reset access.	A real compromised account can pass authentication, expose thread history, create forwarding rules, and support more convincing attacks.
Malware or ransomware entry	A trusted-looking attachment or link starts code execution or credential capture.	The initial email can become lateral movement, data theft, operational shutdown, and recovery work.
Customer and vendor deception	External parties receive fake invoices, payment changes, account notices, or document requests carrying your name.	Even when your systems were not breached, the affected party may associate the fraud with your brand and controls.
Deliverability and reputation damage	Recipients distrust the domain, messages are filtered, or sending sources develop poor reputation.	Sales, password resets, order confirmations, support replies, and billing notices may stop reaching customers reliably.
Privacy, contractual, and notification exposure	Stolen credentials or documents may expose personal, financial, health, customer, or proprietary data.	Obligations depend on the data, contracts, insurance terms, industry, and jurisdictions involved.
Operational and incident-response cost	Staff preserve evidence, reset accounts, revoke sessions, contact banks, notify partners, review logs, and rebuild trust.	The indirect cost can exceed the stolen amount through downtime, legal review, outside specialists, and delayed work.
Executive and board scrutiny	Leadership must explain why a domain was not protected or why a payment could be changed by email alone.	The incident becomes a governance issue, not merely an IT ticket.

A practical remediation roadmap

Phase 1: establish ownership and inventory

- List every registered domain, parked domain, sending subdomain, and brand variation.
- Identify who controls the registrar, authoritative DNS, mail provider, website hosting, applications, and third-party senders.
- Document every system that generates mail, including systems that send only during rare events such as password resets, invoices, alerts, or form submissions.
- Record the business owner, technical owner, vendor contact, sender domain, envelope domain, DKIM domain, selector, and purpose for each source.

Phase 2: make legitimate mail authenticate and align

- Consolidate SPF into one maintainable policy and remain within the DNS lookup limit.
- Enable DKIM for every provider using the company's domain or a deliberately assigned subdomain.

- Correct application and vendor settings so at least one DMARC path aligns with the visible author domain.
- Move bulk, marketing, and externally managed mail to purpose-specific subdomains when separation improves control and reputation management.

Phase 3: collect evidence before enforcing

- Publish a valid DMARC monitoring policy and direct aggregate reports to a protected, monitored reporting destination or analysis platform.
- Compare report sources with the approved sender inventory.
- Investigate unknown sources rather than automatically authorizing them.
- Send representative messages from each legitimate system to multiple providers and inspect the resulting `Authentication-Results` headers.

Phase 4: move from visibility to enforcement

- Use a controlled path from `p=none` to `p=quarantine` and ultimately `p=reject`.
- Protect subdomains deliberately with an appropriate `sp` policy and separate policies where a subdomain has a distinct owner or mail flow.
- Configure parked and non-sending domains with `v=spf1 -all` and DMARC rejection.
- Keep a rollback plan and change log so a delivery problem can be diagnosed without discarding the whole control.

Phase 5: harden inbound mail and the decision process

- Enable spoof intelligence, unauthenticated-sender indicators, first-contact warnings, and executive/domain impersonation protection where the mail platform supports them.
- Require phishing-resistant multifactor authentication where practical and review legacy authentication, forwarding rules, OAuth grants, recovery methods, and administrative roles.
- Never approve bank-account, payroll, credential, or ownership changes solely from an email. Verify through a known phone number or established portal, not contact information supplied in the request.
- Require two-person approval or separation of duties for high-impact payments and vendor-master changes.

Phase 6: operate the control

- Review DMARC reports, provider changes, new applications, and expiring vendor relationships on a defined schedule.
- Make email-sender review part of onboarding and offboarding any SaaS platform, website feature, integration, or marketing tool.
- Test the incident playbook and preserve a method for exporting original messages with complete headers.

What to do when a suspicious payment message has already been acted on

1. **Contact the financial institution immediately** using a trusted number and request the fraud or wire-recall process.
2. **Preserve the original message** in its native format with full headers; do not rely only on screenshots or forwarded copies.
3. **Contain affected accounts** by revoking active sessions and tokens, resetting credentials, reviewing MFA and recovery methods, checking mailbox rules and forwarding, and examining administrative changes.
4. **Verify related transactions and requests** with vendors, employees, customers, and leadership through known communication channels.
5. **Report the event promptly** to the FBI's Internet Crime Complaint Center and follow legal, insurance, contractual, and regulatory guidance appropriate to the organization.

Turn email trust into a managed business control

Sunimod works across websites, WordPress, ecommerce, hosting, custom software, integrations, access, recovery, and ongoing technical care—the same environment where forgotten senders and risky notification workflows often originate.

A Sunimod engagement can trace the systems speaking for your domain, document the approved senders, correct SPF and DKIM alignment, stage DMARC enforcement, review website and application mail, strengthen payment-change workflows, and leave your team with a practical operating record instead of unexplained DNS text.

Ask for an Email Identity and Business Email Compromise Risk Review. Describe your mail provider, website or application platforms, known sending vendors, and the business process you are most concerned about.

Request a quote from Sunimod

Frequently asked questions

Will DMARC stop all phishing?

No. DMARC is designed to validate authorized use of a visible author domain. It is highly relevant to exact-domain spoofing, but a lookalike domain can authenticate its own mail and a compromised real account can pass authentication. Inbound anti-phishing controls, strong account security, and verified business processes are still required.

Is p=none protection?

It is an important discovery state, but it does not request quarantine or rejection of DMARC failures. Treat it as an instrumented transition, not the final control.

Can we publish a second SPF record for another vendor?

No. Authorized sources must be represented in one SPF policy. Multiple `v=spf1` records can cause permanent evaluation errors.

Why might legitimate mail fail after DMARC is enabled?

Common causes include an unknown sending service, a provider signing with its own domain instead of the customer domain, an unaligned envelope sender, content modification that breaks DKIM, forwarding that breaks SPF, or a subdomain with separate behavior. That is why inventory, reporting, and staged enforcement matter.

Should a company publish `p=reject` immediately?

A non-sending or parked domain can often move directly to denial records after ownership is confirmed. An active domain should first identify and align every legitimate sender, observe real results, and prepare a controlled change and rollback plan.

Sources and further reading

1. FBI Internet Crime Complaint Center: 2025 Annual Report
2. RFC 9989: Domain-Based Message Authentication, Reporting, and Conformance
3. RFC 7208: Sender Policy Framework
4. RFC 6376: DomainKeys Identified Mail Signatures
5. CISA Cybersecurity Performance Goals 2.0
6. Microsoft: How SPF, DKIM, and DMARC work together
7. Microsoft: Anti-phishing, spoofing, and impersonation controls
8. FBI Internet Crime Complaint Center

Educational and defensive use: The examples in this article use reserved documentation domains and read-only public DNS checks. They do not provide instructions for sending forged email. Email authentication changes can interrupt legitimate mail when implemented without a complete sender inventory, testing, and rollback plan. Legal, insurance, contractual, and regulatory obligations vary by situation and jurisdiction.