Composition Kills:
A Case Study of Email Sender Authentication

Jianjun Chen[*] Vern Paxson[†*] Jian Jiang[‡]

[*]International Computer Science Institute [†]University of California, Berkeley [‡]Shape Security

Abstract

Component-based software design is a primary engineering approach for building modern software systems. This programming paradigm, however, creates security concerns due to the potential for inconsistent interpretations of messages between different components. In this paper, we leverage such inconsistencies to identify vulnerabilities in email systems. We identify a range of techniques to induce inconsistencies among different components across email servers and clients. We show that these inconsistencies can enable attackers to bypass email authentication to impersonate arbitrary senders, and forge DKIM-signed emails with a legitimate site's signature. Using a combination of manual analysis and black-box testing, we discovered 18 types of evasion exploits and tested them against 10 popular email providers and 19 email clients—all of which proved vulnerable to various attacks. Absent knowledge of our attacks, for many of them even a conscientious security professional using a state-of-the-art email provider service like Gmail cannot with confidence readily determine, when receiving an email, whether it is forged.

1 Introduction

Component-based software design [1] has been widely adopted as a way to manage complexity and improve reusability. The approach divides complex systems into smaller modules that can be independently created and reused in different systems. One then combines these components together to achieve desired functionality. Modern software systems are commonly built using components made by different developers who work independently.

While having wide-ranging benefits, the security research community has recognized that this practice also introduces security concerns. In particular, when faced with crafted adversarial inputs, different components can have inconsistent interpretations when operating on the input in sequence. Attackers can exploit such inconsistencies to bypass security policies and subvert the system's operation.

In this paper, we provide a case study of such composition issues in the context of email (SMTP) sender authentication. SMTP's original design lacked mechanisms to ensure the integrity of the purported sender (and message contents) of an email. To combat email spoofing, modern email servers employ several SMTP extensions—SPF, DKIM, and DMARC—to authenticate the sender's purported identity, as the basis for displaying in email clients assurances of validity to users when they read messages.

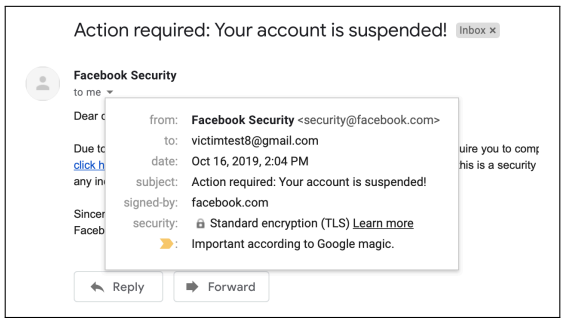


Figure 1: A spoofing example that impersonates `facebook.com`. Gmail shows that this email is signed by `facebook.com`.

We show that the compositions of different software components to construct these validity assurances have wide-ranging vulnerabilities enabling attackers to undermine the decision-making process. Figure 1 illustrates one of our attacks[1] impersonating `facebook.com` on Gmail. The Gmail user sees apparent assurances that the sender was indeed `security@facebook.com` when in fact it was not. Unless otherwise noted, *all* of the attacks we present in this paper manifest similarly: the reader who checks an email for validity receives an apparent-but-incorrect assurance when using a vulnerable email system.

We organize the attacks into three classes. The first class ("*intra-server*") exploits inconsistencies between different components inside a single email server, making the email server generate "pass" authentication results for a spoofed email. The second class ("*UI-mismatch*") exploits inconsistencies between mail servers and the mail clients used to read email, such that the server and the client authenticate/display different email addresses. The third class ("*ambiguous-replay*") replays messages partially protected by DKIM signa-

[1] The A_3 attack, discussed in Section 4.2.

tures, employing additions to yield messages with deceptive contents seemingly signed as authentic by a legitimate site.

We evaluated 10 popular email providers and 19 email clients using a combination of manual analysis and black-box testing. We found 18 types of exploits: 6 of the email providers were affected by *intra-server* attacks, and *all* proved vulnerable to *UI-mismatch* and *ambiguous-replay* attacks.

2 Background

Simple Mail Transfer Protocol (SMTP) provides an Internet standard for mail transmission [2]. Figure 2 shows the three main steps to deliver an email message. Alice's email is first transmitted to her service provider via her mail user agent (MUA). The sending service then sends it to Bob's service provider using SMTP. The message is then delivered to Bob's MUA via IMAP (Internet Message Access Protocol) or POP (Post Office Protocol).

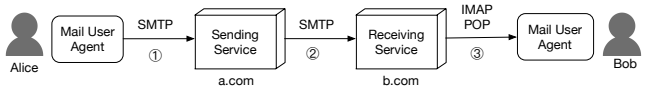


Figure 2: Email transmission from Alice to Bob

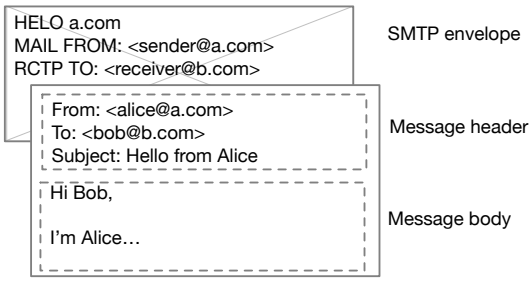


Figure 3: An example of an SMTP message sent from `a.com` to `b.com`.

2.1 SMTP lacks authentication

Figure 3 shows the elements of an SMTP message sent from `a.com` to `b.com`. SMTP's original specification lacked mechanisms to authenticate the sender's identity, enabling any Internet host to impersonate another's identity by sending spoofed emails. In practice, attackers usually exploit SMTP by running their own email servers or clients.

SMTP's design includes multiple "identities" when handling messages. Both the MAIL FROM and From headers identify the email sender, but they have different meanings in an SMTP conversation. The first represents the user who *transmitted* the message, and is usually not displayed to the recipient. The second represents the user who *composed* the message, and is visible to the recipient.

In addition, SMTP introduces multiple other sender identities, such as the HELO command, Sender and Resent-From headers. Nothing in the design enforces consistencies among these. Thus, the design poses a basic question for any authentication mechanism: which identity to authenticate?

2.2 Preventing spoofing with SPF/DKIM/DMARC

To combat email forgery, various email authentication mechanisms have been developed, including SPF [3], DKIM [4], DMARC [5], BIMI [6], and ARC [7]. Our study focuses on the first three mechanisms, as BIMI and ARC haven't been widely adopted yet; we discuss BIMI and ARC in Section 9.

SPF. Sender Policy Framework (SPF) allows a domain owner to publish DNS records to specify which servers are allowed to send emails for the domain. A mail server receiving a message first queries any domain present in the MAIL FROM and—recommended, but not required—HELO commands, to obtain the SPF policy, and then checks whether the sender's IP address matches the policy. If either HELO or MAIL FROM check fails, the mail server enforces the policy specified by domain owner (e.g., hard fail, soft fail) to reject the message.

One major problem of SPF is incompatibility with mail forwarders. When an email is forwarded, SPF checks can fail because SPF components authenticate the forwarding server, rather than the original sending server.

DKIM. DomainKeys Identified Mail (DKIM) uses cryptography to authenticate senders and protect email integrity. The general idea behind DKIM is to let senders sign parts of messages so that receivers can validate them. When sending a message, the sending mail server generates a DKIM-Signature header using its private key and attaches it to the message. When the destination server receives the email, it queries the domain in the `d=` field of the DKIM-Signature header to obtain the signer's public key, and verifies the DKIM signature's validity.

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/
    relaxed; d=example.com; s=selector; h=
    From:To:Subject; l=200;  bh=I8iwjsTG/
    djENwF0HjjQSgUtWKv5izitR9+mDu1ambA=; b=
    HA1a66oMfyVbQwZLd3Dkm3ZDfomVU1FgMF...
```

The above shows an example of a DKIM-Signature header. The important tags for our work include:

- `d` represents the signer's domain.
- `s` stands for selector, which permits multiple keys under the "`d=`" domain for fine-grained signatory control. The tag is used to obtain the public key by querying "s._domainkey.d" (`selector._domainkey.example.com` here).
- `h` represents the list of headers covered by the signature.
- `l` is an optional tag giving the length of the message body covered by the signature.

Unfortunately, neither SPF nor DKIM provides a complete solution for preventing email spoofing. SPF authenticates the HELO/MAIL FROM identifier and DKIM authenticates the d= field in DKIM-signature header: neither of them authenticates the From header displayed to the end-user, which means that even if an email passes SPF and DKIM validation, its From address can still be forged.

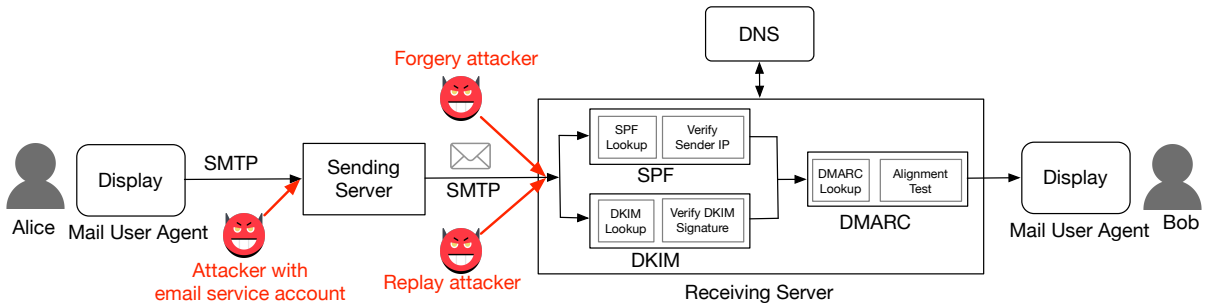


Figure 4: Email authentication flow and three types of attackers.

DMARC. Domain-based Message Authentication, Reporting & Conformance (DMARC) is designed to fix this final trust problem by building on SPF and DKIM. When receiving a message, the receiving mail server queries the domain in the From header to obtain its DMARC policy, which specifies what the receiver should do with the incoming email. The receiving server performs an *identifier alignment test* to check whether the domain in the From header matches the domain name verified by SPF or DKIM. The alignment test has two modes: *strict* and *relaxed*. In strict mode, the From header domain needs to exactly match the SPF or DKIM-authenticated identifier. In relaxed mode (default mode), it only need to have the same registered domain [8]. If either SPF or DKIM indicates a positive result, and the From domain passes the alignment test, the email passes DMARC authentication. This design provides more robustness, for example, for forwarded emails: SPF may fail, but DKIM will survive. If both fail, the server will enforce the DMARC policy specified by the domain owners, such as rejecting the email and sending failure reports.

Combining these three mechanisms, an email system ensures that the address in the From header cannot be forged, and prevents email forgery.

2.3 Email processing flow

Figure 4 shows the main components in the email processing flow. An email sent by a Sending Server goes through two phases before reaching the end-user recipient: authentication by the Receiving Server, and display by the mail user agent (MUA). In the first phase, the Receiving Server verifies whether the email was indeed sent by the purported address, as outlined in the previous section. If the email passes the DMARC verification, it enters the user's inbox.

In the second phase, the MUA (e.g., local mail clients and web interfaces) parses the authenticated email and displays the message to the end-user recipient, including, potentially, an attestation of the sender's identity. Although authenticated emails include different sender identities in their headers—such as From headers, MAIL FROM (aka Return-Path) and DKIM-Signature headers, usually the MUA only displays the From header as the message sender. Thus, the From header provides the key identity relevant for gaining the user's trust, and as such merits particular protection.

3 Composition challenges in email authentication

We now turn to an analysis of how the composition of different processing components in the email delivery and presentation chain can lead to an array of vulnerabilities that undermine sender authentication.

3.1 Threat model

We consider three types of spoofing attackers: forgery attackers, replay attackers, and attackers who have accounts on legitimate email services.

A *forgery attacker* can send arbitrary emails to victims (`victim@victim.com`) directly from their mail server (`attack.com`). The attacker spoofs the email's sender in the From header to a legitimate website's address (`admin@legitimate.com`), which—nominally—email authentication mechanisms should prevent.

Replay attackers possess emails with valid DKIM signatures signed by a legitimate website domain. These attackers exploit modifications to email headers, and potentially the email body, that will not break DKIM signatures. These attackers can obtain such DKIM-signed emails from, for example, advertisement emails, registration emails, or public mailing lists.

Malicious users of legitimate email providers exploit the failure of some email providers to perform sufficient validation of emails received from local MUAs. These attackers can send emails with spoofed From headers. The exploited email providers may automatically attach DKIM signatures to their outgoing emails, enabling the attackers to impersonate other users of the email provider.

In this work we assume that 1) the targeted legitimate sites configure SPF/DKIM/DMARC mechanisms correctly, and 2) the target email services reject emails that fail DMARC authentication. In such a deployment environment, an email authentication system should prevent spoofed email from ever passing the authentication tests, ensuring that end-users always see authenticated email sender addresses.

Security requirement. To achieve this goal, an email system should provide the following basic security requirement: The end-user Bob who uses email client C to receive an email from receiving server R can determine that the message is

indeed from user Alice of sending server S, if and only if: (1) The From header of the email that S sends matches the authenticated username (other users of S cannot spoof Alice's address); (2) SPF/DKIM/DMARC components in R can obtain S's DNS correct policy; (3) SPF/DKIM and DMARC components in R consistently authenticate the same identifier; (4) the identifier that R authenticates is consistent with the identifier that C shows to Bob.

Challenges in preserving the requirement. This requirement, although intuitive, implies a set of semantic binding relations that every component in the email processing chain must respect. Doing so turns out to pose considerable challenges, particularly for decentralized systems with different components built by different developers. These include:

1) The difficulty of coordinating across components. Although standards exist to ensure that different components behave in predictable ways, standards documents often provide vague implicit descriptions open to different interpretations. For example, when DMARC leverages SPF to prevent email spoofing, the DMARC component might assume that the SPF component always authenticates the MAIL FROM identifier if the MAIL FROM address is not empty; but SPF does not provide this guarantee. The SPF component might forward HELO authentication results and leave to DMARC to itself check which identity is verified. As a consequence, DMARC and SPF components authenticate different identifiers, leading to email authentication bypass (per Section 4.1).

2) Tensions with the robust principle. Postel's Law encourages implementations to be permissive in how they process malformed inputs. Although doing can significantly facilitate connectivity between trusted parties, in an adversarial context it can also introduce exploitable ambiguities. As we show in Section 5.1, different preferences on tolerating malformed From headers between mail servers and email clients can lead to numerous email spoofing attacks.

3) The danger of feature composition. Implementations can vary in supporting various features, such as protocol extensions or older versions, or customizable functionality. Such diverse behavior appears harmless when examining each component independently, but can in combination introduce security problems. Attackers can chain different feature gadgets across components to perform unexpected computation. As we show in Section 5, different combinations of email providers and clients can suffer from vulnerabilities simply because they differ in their support for various features.

3.2 Testing methodology

To investigate how real-world systems handle these challenges, we conducted a security analysis of popular email providers and MUAs.

Selecting email providers and clients. We chose to test email providers that 1) verify SPF/DKIM/DMARC for incoming email, 2) allow us to register accounts for testing, and 3) reflect SPF/DKIM/DMARC authentication results in the email headers. For MUAs, we gathered a list of popular local email clients[2] that covers today's major platforms. We also tested the web interfaces of selected email providers by using their third-party email importation functions. In total, we tested 10 email providers and 19 MUAs, including 9 local email clients and 10 web interfaces, as shown in Table 2.

Black-box testing. The problems we examine are rooted in the inconsistent behaviors of different programs. Our analysis followed a behavior-oriented methodology that dissects an email authentication workflow, dividing it into four steps.

First, we studied SMTP and email specifications (both core protocols and extensions), extracting authentication-related behavior, focusing on the lexical, syntax and semantic rules for different identities. Second, we generated ambiguous test cases by "walking" through the extracted rules to examine each of their choice points, in a manner analogous to that employed in prior work for finding IDS evasion threats [9]. Third, we leveraged the generated cases to test different components for inconsistent behaviors in parsing and interpreting ambiguous messages. Finally, we manually analyzed the identified behaviors to verify the likelihood of success in practice.

We define an email authentication mechanism as *broken* when the following both hold: 1) the email server erroneously verifies the test email's sender as not spoofed, for example, DMARC authentication produces a "pass" result; 2) the MUA erroneously indicates that the sender address is from a (legitimate) target domain rather than the attacker's sending domain.

To extend our results to closed-source proprietary systems, we first examined popular open-source SMTP implementations,[3] to understand their possible interactions and find potential ambiguities. Guided by these results, we then probed the possible internal logic of black-box systems, testing any discovered ambiguities to assess whether they reflect similar vulnerabilities.

Leveraging this approach, we found three categories of attacks leading to "broken" authentication results: *intra-server*, *UI-mismatch*, and *ambiguous-replay* attacks. *Intra-server* attacks exploit ambiguities between an email server's different internal components. *UI-mismatch* attacks exploit inconsistent interpretations between mail servers and MUAs. *Ambiguous-replay* attacks produce misleading DKIM-signed emails that validate as signed by a (legitimate) target domain. Tables 1 and 2 below summarize the susceptibility of the different email providers and MUA clients that we studied. While 4 of the 10 email providers resist *intra-server* attacks, all have vulnerabilities to *UI-mismatch* and *ambiguous-replay* attacks.

We now detail how we explored the three attack categories, illustrated with representative cases.

[2] Mainly from https://emailclientmarketshare.com/.

[3] Postfix, Python-postfix-policyd-spf, OpenDKIM, and OpenDMARC.

4 Intra-server Attacks

Intra-server attacks exploit inconsistencies between different internal components of a single implementation. Per Figure 4 above, sender authentication can include four internal components: SPF, DKIM, DMARC, and DNS. We discovered three techniques to exploit their inconsistencies: (1) HELO/MAIL FROM confusion (A_1, A_2); (2) ambiguous domains (A_3); and (3) authentication results injection (A_4, A_5).

Table 1: Email providers vulnerable to Intra-server attacks.

Email Providers	Ambiguity b/w SPF&DMARC	Ambiguity b/w DKIM&DNS	Ambiguity b/w DKIM&DMARC
Gmail.com		✓(A_3)	
iCloud.com	✓(A_5)		✓(A_4)
Outlook.com			
Yahoo.com			
Naver.com			✓(A_4)
Fastmail.com			
Zoho.com	✓(A_5)		✓(A_4)
Tutanota.com	✓(A_2, A_5)		✓(A_4)
Protonmail.com	✓(A_5)		✓(A_4)
Mail.ru			

"✓": vulnerable to specific attack(s) due to internal inconsistencies.

4.1 HELO/MAIL FROM confusion

SMTP employs two different identifiers—HELO and MAIL FROM—to represent the email sender who transmits a message. The SPF standard (RFC 7208) states that SPF verifiers should check both; checking MAIL FROM is mandatory, and HELO is recommended. The DMARC standard (RFC 7489) states that DMARC verifiers should use the MAIL FROM identity to perform the *alignment test* to validate the identity in the From header. If the MAIL FROM address is empty, the verifier should use the HELO identity.

This design introduces the possibility that different components might authenticate different identifiers. When the SPF component cannot verify the MAIL FROM address, but can verify the HELO identifier, the DMARC component might still use the MAIL FROM identifier for its alignment test. We developed two techniques to exploit these possibilities:

1) Non-existent subdomains (A_1). The first technique crafts a MAIL FROM domain as a non-existent subdomain of a legitimate domain, as shown in Figure 5a. SPF components cannot verify the MAIL FROM address because the non-existent domain doesn't have any SPF policy. Some SPF implementations (e.g., Python-postfix-policyd-spf) will then only verify the HELO identifier, forwarding a "pass" result because the HELO domain is under the attacker's control. Some DMARC implementations (e.g., OpenDMARC), however, still use the MAIL FROM domain to perform the alignment test with the From header, because the MAIL FROM address is not empty. Doing so subverts the DMARC authentication because both the SPF check and the DMARC alignment test show positive results.

2) "Empty" MAIL FROM addresses (A_2). The second technique exploits differences in how components treat an empty MAIL FROM address, per Figure 5b. (Note that in the example, the left parenthesis is deliberately left unclosed.) Some SPF implementations treat "`(any@legitimate.com`" as an empty MAIL FROM address, and thus forward the results of checking HELO to the DMARC component, because the string in the parentheses can be parsed as a comment according to RFC 5322 [10]. Some DMARC implementations, however, may take it as a normal non-empty address, and use its domain for the alignment test.

4.2 Ambiguous domains

Inconsistencies can also arise between authentication components and DNS components: what the authentication component verifies differs from what the DNS component queries. An attacker can craft ambiguous domains to make the authentication component believe that it's verifying the legitimate domain, but the DNS component actually queries the attacker's domain to obtain policy records. The authentication component generates "pass" authentication results because the attacker controls the policy retrieved via DNS.

NUL ambiguity (A_3). One way to craft such domains uses the NUL ("\x00") character, which terminate strings in some languages (e.g., C) but not in others (e.g., Perl or PHP). For example, we can fool Gmail.com using this technique. Gmail's DKIM and DNS components differ in interpreting NULs in domain name, which we exploited for our example in the Introduction (Figure 1).

Per Figure 5c, first the attacker constructs a fake email with arbitrary email content. They then sign the message with their own private DKIM key to generate the DKIM-Signature header, which specifies the "d=" tag as `legitimate.com` and the 's=' tag as "`attacker.com.\x00.any`".

When the Gmail server receives the email, its DKIM component queries the domain *s._domainkey.d*, i.e., "`attack.com.\x00.any._domainkey.legitimate.com`", to obtain the public key. But when it invokes to resolve this domain, the DNS component parses the NUL character as a string terminator and instead obtains the public key from `attack.com`. The DKIM component thus uses the attacker's public key to verify the forged message, erroneously believing that the legitimate domain correctly signed the message. The spoofed message also passes Gmail's DMARC verification because the "d=" domain is identical to the From header domain.

4.3 Authentication results injection

Another vector for potential ambiguity arises from how results are communicated from one component to another. The presence of meta-characters in the communication introduces the possibility of "results injection" analogous to SQL or command injection.

Authentication result header syntax. This threat depends on the details of how SPF and DKIM components forward their authentication results to DMARC components to enable it to perform its alignment check on the value of the From header. RFC 8601 defines the Authentication-Results header to provide a common framework for communicating these

(a) HELO/MAIL FROM confusion. (b) "Empty" MAIL FROM address. (c) NUL ambiguity.

(d) DKIM authentication results injection. (e) SPF authentication results injection #1. (f) SPF authentication results injection #2.

Figure 5: Different *intra-server* attacks to make SPF/DKIM verify `attack.com` while DMARC instead uses `legitimate.com`.

authentication results, for example:

```
Authentication-Results: example.com; spf=pass
    smtp.mailfrom=sender@sender.com; dkim=
    pass (1024-bit key) reason="signature ok"
     header.d=sender.com;
```

Here, "`spf=pass`" and "`dkim=pass`" indicate that the message passed both SPF and DKIM verification for the mail server for `example.com`. "`smtp.mailfrom`" represents the domain verified by the SPF component, and "`header.d`" represents the domain verified by the DKIM component. The text in parentheses reflect a comment. The DMARC component parses this header to extract the SPF/DKIM authentication results and check whether the tested value align with the domain in the From header.

Authentication results injection attacks. A vulnerability arises because an attacker can control the domain name embedded in the "`header.d`" and "`smtp.mailfrom`" fields. The flexibility of domain-name syntax provides fertile ground for attackers to construct malformed domain names. Although many applications require domain names to follow specific syntax rules—for example, domain name registrars only allow users to register domain names under the LDH rules (only letters, digits, hyphens)—the DNS protocol does not impose any restrictions on the characters in a domain label.

In particular, an attacker can introduce malformed domains that include meta-characters, for example "`a.com(.b.com`". SPF and DKIM components may treat those characters as data, while DMARC components may parse them as control information. We found two types of injection attacks based on such malformed domains.

1) DKIM authentication results injection (A_4). Per Figure 5d, attackers can generate DKIM-Signature headers using their own private keys, with "`d=`" values that embed a literal open parenthesis, such as "`legitimate.com(.attacker.com`".

When receiving this message, the DKIM component queries "`selector._domainkey.legitimate.com(.attacker.com`"—a domain under the attacker's control—to obtain the DKIM public key to verify the message. The DKIM component then generates:

```
Authentication-results: victim.com; dkim=pass
    (1024-bit key) header.d=legitimate.com(.
    attacker.com;
```

When receiving the Authentication-Results header, the DMARC component parses "header.d" as `legitimate.com`, because it parses the content after the "`(`" as a comment. Since the "`header.d`" value matches the From header domain, the attacker's message passes DMARC verification.

Along with "`(`", double (`"`) and single (`'`) quote characters can also work for this technique. Because RFC 5322 defines characters within the quotes as atoms, DMARC modules may parse the content after the quote as part of the atom.

2) SPF authentication results injection (A_5). Similarly, an attacker can craft malformed addresses in MAIL FROM commands to bypass SPF and DMARC verification, as shown in Figure 5e. THe SPF component verifies the attacker-controlled domain "`legitimate.com(.attacker.com`", while the DMARC module takes the first half of the domain for the alignment test.

We found that some mail servers perform a degree of validation on the MAIL FROM address's syntax, and reject the above address. But attackers can bypass their validation as shown in Figure 5f. Here, the mail server takes the second "`@`" as the delimiter, and recognizes it as a valid email address, while the SPF component takes the first "`@`" as the delimiter,

and thus queries "`legitimate.com'@a.attack.com`"—the attacker's domain—to verify the sending IP address. When the DMARC component parses the authentication results, it takes the content after the single quote as a quoted string, and uses `legitimate.com` for the alignment test.

5 UI-mismatch Attacks

As shown in Figure 4, email servers and mail user agents (MUAs) process messages separately. *UI-mismatch* attacks exploit the inconsistencies between how an email server validates a message versus how the MUA ultimately indicates its validity. Generally, we can divide From header-related processing into two phases: 1) parsing a MIME message to extract the From header; 2) parsing the From header to extract a corresponding domain or email address. We likewise divide our *UI-mismatch* attacks into two categories: *ambiguous From headers* and *ambiguous email addresses*.

5.1 Ambiguous From headers

We devised three techniques to exploit ambiguous From headers: 1) multiple From headers; 2) space-surrounded From headers; 3) From alternative headers.

1) Multiple From headers (A_6). RFC 5322 states that an email message must have exactly one From header, which implies that email messages with multiple From headers are invalid and should be rejected by receiving services.

We find that 19 out of 29 tested implementations (including 5 email providers and 14 MUAs) do not in fact follow the specification and reject such messages. All 5 email providers use the first From header for DMARC checking. iCloud.com (Web) and Mail (Windows) display the last From header; Mail (MacOS) shows both headers; and the other 11 MUAs display the first From header.

Thus, attackers can mislead the presentation to the user of email sender validity by using a mail server that (1) accepts multiple From headers, (2) with a different preference than the user's email client. Figure 6a shows such an example. iCloud (Server) uses the first From header for DMARC verification, but iCloud (Web) displays the second one to the user.

2) Space-surrounded From headers (A_7). RFC 5322 defines an email header as a field name, a colon, and a field body (value). If an attacker violates this syntax structure by inserting whitespace before or after the header name, different implementations handle the ill-formed header differently.

We identify three such edge cases: a) a space-preceded From header as the first header; b) a space-succeeded From header; c) a folding-space-succeeded From header. The email standards implicitly disallow the first two cases, and explicitly disallow the last case. In practice, none of our tested implementations fully comply with the specification. Protonmail.com (Server) rejects the first and second case, Yahoo.com (Server) rejects the third case. Others recognize the space-surrounded From header as a valid From header, take it as an unknown header or parse the whitespace as the delimiter between email headers and body.

Whitespaces open new opportunities for multiple From ambiguities. First, use of whitespace can bypass the email server's validation. For example, Mail.ru (Server) rejects email with multiple From headers, but an attacker can bypass it with a folding-space-succeeded From header, as shown in Figure 6b. Second, inconsistent interpretation of whitespace can lead to ambiguities. Mail.ru (Server)'s DMARC component recognizes the folding-space-succeeded From header and authenticates `attack.com`, but Outlook (Windows) takes it as an unknown header and presents `admin@legitimate.com` as the validated From header.

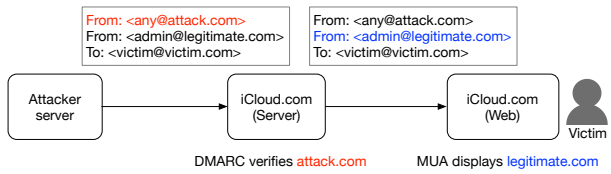
Sometimes we can even fool the email servers and MUAs that use the same header parsing and processing, by leveraging special forwarding behaviors of the email servers. Figure 6c shows an example. Both Fastmail.com (Server) and Fastmail.com (Web) don't recognize the space-succeeded From header, but Fastmail.com (Server) *normalizes* the space-succeeded From header, removing the space when forwarding the message. The forwarding behavior causes Fastmail.com (Web) to recognize a different From header.

3) From alternative headers (A_8). RFC 5322 includes multiple headers that identify different email sender roles. The From header represents the user who writes the message, the Sender header the user who submits it, and the Resent-From header the user who forwards the message.

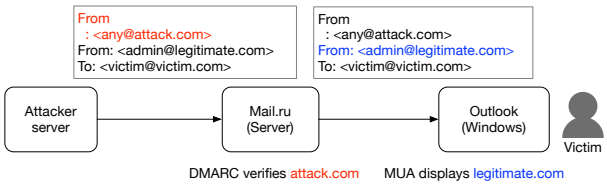
Normally, only the From header plays a role in email authentication and display. However, if an attacker crafts an email with no From header or an unrecognized From header, some implementations will use alternative headers to identify the message sender. We found 7 out of 19 MUAs have such behavior. Gmail.com (Web) shows the Resent-From header value when the From header is missing; the other 6 display the Sender header value in the From field. All of the email servers we tested only use the From header for DMARC verification. If From header is not found, they don't perform DMARC authentication, or generate "none" results.

The interplay between From header and its alternative headers introduces another source of ambiguity. As shown in Figure 6d, Naver.com (Server) recognizes a folding-space-succeeded From header and verifies `attack.com`, but Outlook (Windows) doesn't recognize it and shows the (unverified) Sender header value in the From field.

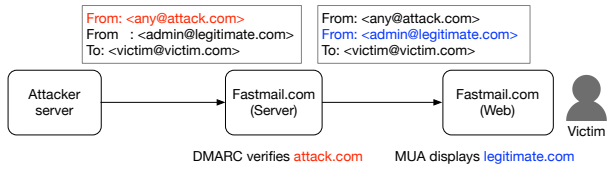
Attackers can also combine different techniques to chain multiple features to bypass strict security validation. Figure 6f shows an example. Gmail.com (Server) has strict message validation: it rejects messages with multiple From headers, and adds a new From header with the MAIL FROM value if the From header is absent. But an attacker can bypass this validation by combining a space-preceded From header as the first header, a Resent-From header as an alternative header, and empty MAIL FROM value. Gmail.com (Server) recognizes the first space-preceded From header and uses it to perform DMARC checks. It then inserts an Authentication-results header before the message, which causes the original From



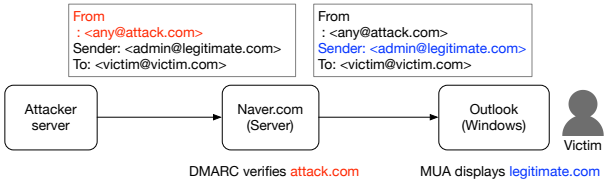
(a) Preference of multiple From headers.



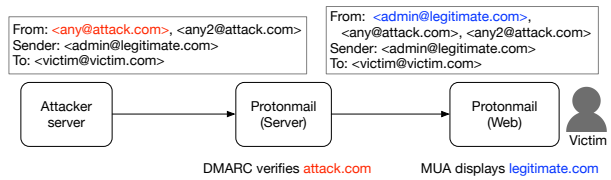
(b) Inconsistent interpretation in folding-space-succeeded From header.



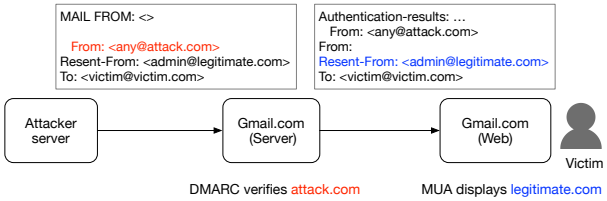
(c) Exploiting normalization behavior with space-succeeded From header.



(d) Interpreting Sender header as From alternative header.



(e) Exploiting normalization behavior with multiple email addresses.



(f) Combine multiple techniques to bypass Gmail validation.

Figure 6: Different cases of inconsistent interpretation of From header between email servers and MUAs.

header to be parsed as a "folded" line, i.e., a continuation of the Authentication-results header. It then adds a new From header with the empty MAIL FROM value and forwards the message to the email client. Gmail.com (Web) ignores the empty From header, and displays the Resent-From header value as the message sender.

Figure 7: An example of valid From header.

5.2 Ambiguous email addresses

Even if an email server and client extract the same From header from a MIME message, extracting a *consistent* email address from that From header poses another challenge due to the complex syntax of From headers. In this section we develop a set of attacks that exploit these complexities.

Complex From header syntax. Figure 7 shows a valid From header with a single mailbox address, which consists of four elements.

Display name is an optional field that identifies the sender's name. As this field is not protected by SPF, DKIM or DMARC, many known phishing attacks use the display name to deceive victims. (In this paper, however, we aim to spoof the real address, rather than the display name.)

Real address indicates the real sender. It consists of a local-part, "@", and a domain. The local part can be a string with or without quotes.

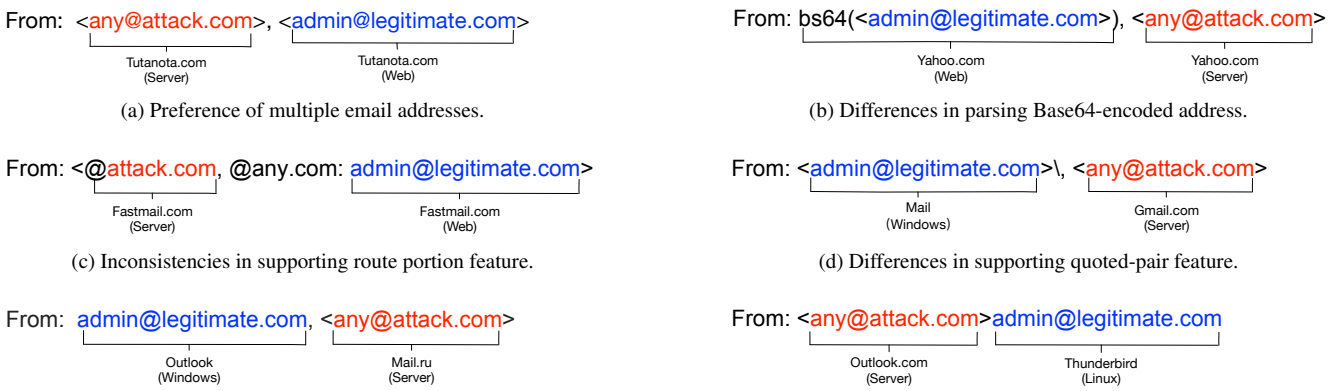
Route portion is an obsolete feature originally defined in RFC 822 to indicate the delivery path that the message should follow. Its syntax is a comma-separated list of domain names, each preceded by "@", with the list terminated by a colon. RFC 5322 prohibits *generating* this obsolete field, but recipients still must *accept* it (and ignore the routing part).

Comments is a string of characters enclosed in parentheses that provide some human-readable information. Comments can be freely inserted in many places of a From header, such as before or after the address, or inside the real address. For example, RFC 5322 Appendix A.5 states that "`From: Pete(A nice \) chap) <pete(his account)@silly.test(his host)>`" is a valid address.

Multiple address lists. RFC 5322 specifies that the From header value can be a mailbox address list, which indicates that the message has multiple authors. This means that addresses such as that one in Figure 7 can be repeated multiple times, separated by commas. The RFC also states that if the From header has multiple addresses, a Sender header with a single mailbox address must appear in the message.

Quoted-pair. RFC 5322 reserves some characters for special interpretation, such as commas and quotes. To permit the use of these characters as uninterpreted data, email senders can use '\' to escape them.

Encoding. Originally SMTP only allowed US-ASCII characters in email headers. To support non-ASCII characters, RFC 2047 defined two encoding approaches: Base64 en-



(a) Preference of multiple email addresses.

(b) Differences in parsing Base64-encoded address.

(c) Inconsistencies in supporting route portion feature.

(d) Differences in supporting quoted-pair feature.

(e) Inconsistencies in recognizing special character precedence.

(f) Display name and real address parsing ambiguity.

Figure 8: Different cases of inconsistent interpretations of email addresses between email servers and MUAs.

coding and quoted-printable encoding. Its syntax is like this: `=?charset?encoding?encoded-text?=`, in which the "charset" field specifies the character set of the unencoded text;"encoding" value should be "B" or "Q", representing the encoding algorithm; "encoded-text" is the text encoded by the algorithm. For example,"`From: bob<b@b.com>`" can be encoded as "`From: =?utf-8?B?Ym9i?=<b@b.com>`" by the Base64 encoding approach.

Attacks leveraging complex From headers. We find that implementations vary in parsing and interpreting From headers. Here we show five attacks that exploit these inconsistencies, as shown in Figure 8.

1) Multiple email addresses (A_9). We observe 5 distinct behaviors in processing From headers listing multiple addresses. Gmail.com (Server) and Mail.ru (Server) reject the messages; Tutanota.com (Web) displays the last address; Zoho.com (Server) and iCloud.com (Web) don't verify or display any address; 2 mail servers and 4 MUAs verify or display all of the addresses; all the others take the first address.

Multiple email addresses enable two new kinds of ambiguities. First, when the mail server rewrites addresses in From headers (for example, Protonmail.com (Server) inserts the Sender address into the From header), the mail server may recognize a From header value that differs from the email address that the client displays, as shown in Figure 6e. Second, if the mail server forwards the From header as-is, different interpretations of multiple email addresses can directly lead to authentication bypasses. In Figure 8a, Tutanota.com (Server) only uses the first address for DMARC checking, while its web interface only shows the second one.

2) Email address encoding (A_{10}). Figure 8b shows an example exploiting the differences in parsing encoded addresses. In our experiments, Yahoo.com (Server), Outlook.com (Server), iCloud.com (Server), Fastmail (Server), Zoho.com (Server) and Tutanota.com (Server) don't recognize the encoded address, and use `attack.com` for DMARC testing; but Gmail.com (Web), Outlook.com (Web), Yahoo.com (Web), Naver.com (Web), Mail (MacOS), Mail (Windows) and Mail (iOS) support this encoding feature, and only display the first address.

3) Route portion (A_{11}). As shown in Figure 8c, Fastmail.com (Server) does not recognize the route portion, and treats `attack.com` as a real address to use for DMARC verification; while 10 MUAs, including Fastmail.com (Web), ignore the route portion, and only show `admin@legitimate.com`.

4) Quoted-pairs (A_{12}). Figure 8d shows an example arising from differences in supporting the quoted-pair feature. Gmail.com (Server) and iCloud.com (Web) recognize the second address; but Mail (Windows), iCloud (Server) and 12 other implementations only use the first one.

5) Parsing inconsistencies (A_{13}). We also found inconsistencies in recognizing the precedence of different delimiters. Figure 8e shows an example. Mail.ru (Server) and Zoho.com (Server) DMARC component believes that "<" has higher priority, and authenticate `attack.com`; but Outlook (Windows) and 8 other MUAs have a different preference, and only display `legitimate.com`.

Differences in parsing display names and real addresses provide another source of ambiguity. As shown in Figure 8f, Thunderbird (Linux), Mail.ru (Web), Gmail.com (Server) and Mail.ru (Server) mistakenly validate or display `admin@legitimate.com` as the real sender but Outlook.com (Server), iCloud.com (Server), Protonmail.com (Server) and 9 other implementations recognize it as `attack.com`.

Broader issues. SPF, DKIM, and DMARC rely on domain queries for sender authentication. When failing to obtain the domain record, the mail service providers may decide that the domain doesn't deploy the corresponding security mechanisms, and allow the message into the user's inbox. Leveraging this "fail-open" feature, an attacker can further exploit inconsistencies between mail servers and MUAs to bypass authentication. Here are three examples:

1) Invisible characters (B_1). An attacker can by-

Table 2: Vulnerability of the tested email providers and MUAs to UI-mismatch attacks.

Servers \ MUAs	Web interface	Windows		MacOS		Linux	Android		iOS	
		Mail	Outlook	Mail	eM Client	Thunderbird	Gmail	Outlook	Mail	Gmail
Gmail.com	✓	✓			✓			✓		✓
iCloud.com	✓	✓				✓				
Outlook.com	✓					✓				
Yahoo.com	✓									✓
Naver.com	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fastmail.com	✓	✓			✓			✓		✓
Zoho.com		✓	✓		✓			✓		✓
Tutanota.com	✓	—	—	—	—	—	—	—	—	—
Protonmail.com	✓	—	—	—	—	—	—	—	—	—
Mail.ru		✓	✓	✓	✓	✓	✓	✓	✓	✓

"✓": email server and MUA combination where we can expose an inconsistent interpretation.
"—": email providers that don't support third-party MUAs for our testing account.

pass Outlook.com authentication by appending invisible characters to the target domain, for example, "From: `admin@legitimate.com\u2000`". The DMARC module in Outlook.com (Server) treats `legitimate.com\u2000`" as a new domain and doesn't locate any policy for it, while its web interface only shows `legitimate.com`.

2) Encoding (B_2). When an attacker sends a From header with Base64-encoded email address, e.g., "From: *base64encode*(`admin@legitimate.com`)", the DMARC module of Yahoo.com (Server) authenticates the encoded domain, but its web interface shows the decoded address.

3) From alternative headers (B_3). Upon receiving a message that has no From header but does have a Sender header, Outlook.com (Server), Zoho.com (Server), and Tutanota.com (Server) omit DMARC verification or generate "none" results for the message. However, their web interfaces show the Sender header value.

6 Ambiguous-replay Attacks

Attackers can also spoof emails with seemingly valid DKIM signatures from legitimate domains, bypassing both DKIM and DMARC authentication safeties to make forged emails more deceptive.

DKIM uses digital, cryptographic approaches to prevent tampering with signed content. However, two DKIM mechanisms make signature spoofing possible. First, DKIM doesn't prevent replay attacks. A replay attacker who has an email signed by a legitimate domain can resend it to other victims, a known issue already noted in the DKIM standard. Second, DKIM allows attackers to append additional email headers—*or even body contents*, in some cases—to the original message. Combining these two weaknesses, a replay attacker can append malicious content without breaking the DKIM signature, and further fool email clients to only display the attacker's content by exploiting inconsistencies between DKIM processing and MUA presentations.

6.1 DKIM signature replay attacks

As mentioned in Section 2, DKIM signatures protect both email headers and bodies. The latter is always signed. Signing headers, however, is optional, and specified by the "`h=`" tag of the DKIM-Signature header.

1) Header spoofing (A_{14} and A_{15}). We found two techniques to spoof email headers. First (A_{14}), if the headers in the "`h=`" tag are incomplete, a replay attacker can modify those unprotected headers and send the result to other victims. RFC 6376 lists 19 headers which should be signed, including From, Subject, To and Content-Type. Among them, however, only the From header must be signed; the others are recommended options. In real-world deployment, different sites have various choices. For example, `citibank.com` only signs "`h=from:subject`" headers; `americanexpress.com` only signs "`h=from;reply-to`"; `aa.com` (American Airlines) only signs "`h=from`". A replay attacker can modify these unprotected fields in signed messages without invalidating DKIM signatures. Figure 9 shows a spoofing example of exploiting American Airlines DKIM signatures. The attacker can make Gmail.com render the original body as an attachment, by setting the Content-Disposition header to be "`attachment;filename=ticket.jpg`".

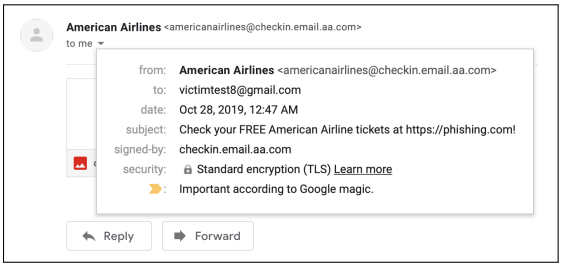


Figure 9: An example of replaying an American Airlines email to a Gmail.com recipient. The subject is fake and the original body is rendered as an attachment.

Second (A_{15}), while including all necessary headers in the signature can prevent attackers from tampering them, a replay

attacker can still bypass the checks by using multiple headers, per Section 5.1. An attacker can craft ambiguous emails by adding a new header (e.g., Subject) to the signed mail, if two parties in the email process chain parse and interpret the extra header differently; for example, if the DKIM component uses the original Subject header while the mail client uses the crafted Subject header.

While RFC 6376 § 5.4.2 states DKIM components must use the last header if a message has duplicate headers, we find that DKIM components and email clients indeed sometimes lack consistency in processing multiple headers. In our testing experiments, all tested DKIM components conformed with the rule—but 10 out of 19 MUAs prefer the first header.

```
DKIM-Signature: v=1; a=rsa-sha256; q=dns/txt;
     c=simple/relaxed; s=default;
     d=service.discover.com;
     h=From:Sender:To:Subject; l=200;
     bh=z61ep91pq...; b=aPg+UnM+wYY7T784XRM+bQ...
From: Discover Card <discover@service.discover.com>
To: victim@victim.com
To: any@any.com
Message-ID: <1518338104553@discoverfinancial.com>
Subject: Action required: Your account is suspended!
Subject: Your statement is available online
Content-Type: multipart/mixed; boundary=BAD
Content-Type: text/plain; charset=UTF-8

Dear customer,

Your bank statement is available online...
--BAD
Content-type: text/plain

Dear customer,

Your account is suspended...

Thanks,
--BAD--
```

Figure 10: An example of exploiting a discover.com DKIM signature to a Gmail.com recipient.

2) Body spoofing (A_{16}). Apart from spoofing the email header, an attacker can also spoof the email body by exploiting the optional "l=" tag in the DKIM-Signature header, which represents the length of the email body included in the signature. This tag is intended for increasing signature robustness when sending to mailing lists that modify email body content. For example, Google Groups usually appends unsubscribe information at the end of each forwarded email. Such behavior can break DKIM validations.

Use of "l=" allows a replaying attacker to append new malicious contents to the original email body without breaking the DKIM signature. In addition, if the Content-Type header is not protected by the DKIM signature, the attacker can further change the email MIME structure by redefining it so that mail clients only display the attacker's malicious content.

Figure 10 shows an example spoofing a discover.com email to a Gmail.com recipient. The red part shows the newly crafted content. As discover.com uses "l=" tag in its signature, and the Gmail server takes the *last* instance of duplicate headers for DKIM verification, the crafted email passes Gmail's DKIM validation. When the Gmail web interface displays this message, it uses the MIME boundary defined by the attacker and only shows the attacker's content, because RFC 2046 § 5.1.1 specifies that any content before the boundary is treated as preamble and not displayed by email clients.

We conducted a preliminary assessment of this problem by collecting emails from wikileaks.org, IETF mailing lists, and our personal emails. We find that many sites are not aware of this attack. Among the 10 email providers we tested, Zoho.com includes the vulnerable "l=" tag for its outgoing messages. Popular sites such as baidu.com, discover.com, akamai.com, manuscriptcentral.com, badoo.com (Alexa 803), and blizzard.com (Alexa 1,066) are also vulnerable to this technique.

6.2 Spoofing via an email service account

An attacker can also leverage access to an email service to spoof misleading DKIM-signed emails. In this scenario, the attacker has an account on a legitimate email service, but uses a custom MUA to originate emails sent through the service. Email providers will first authenticate the MUA using the username/password provided in the AUTH command. They will then check whether the From header in the message matches the authenticated username. If so, the email provider attaches its DKIM signature when forwarding the message.

The problem (A_{17}) arises when an email provider does not perform sufficient checks on the From header, enabling an attacker to send a signed message with another user's address (e.g., administrator). As the message has the email provider's DKIM signature attached, it will pass the receiver's DKIM and DMARC validation.

Given the complexity of From header syntax, its validation is difficult and error-prone. An attacker can use the techniques described in Section 5, such as ambiguous From headers and email addresses, to bypass the email provider's validation.

Of the 8 email providers we tested,[4] all except Outlook.com are vulnerable to this attack. Fastmail.com (Server) accepts arbitrary email addresses in the From header, even email addresses from different domains. iCloud.com (Server), Naver.com (Server) and Zoho.com (Server) accept multiple From headers and only check if the first one matches with the authenticated username. Yahoo.com (Server), iCloud.com (Server) and Naver.com (Server) accept multiple addresses and only check the first address. Gmail.com (Server), Zoho.com (Server), mail.ru (Server) accept From headers like "From:admin@a.com\,<user@a.com>" and only check the second one. The message will pass the receiving server's DKIM and DMARC validation, while email clients may display the unverified (e.g., administrator) address, as presented in Section 5.

[4] We omit Tutanota.com and Protonmail.com as they do not support third-party MUAs for our testing account.

6.3 Replay attacks to subvert DKIM signatures

An attacker with an account on an email service can also employ replay attacks to forge DKIM-signed emails even for email providers that perform strict From header validation, such as Outlook.com.

The spoofing attack (A_{18}) proceeds in two steps. First, the attacker uses their account to email themselves through the email provider server. In the email, the attacker can create deceptive content in the email body, Subject header and To header, but not the From header given the email providers strict validation. When the email provider sends the message, it attaches its DKIM signature to the message.

Second, the attacker adds an extra From header with another user's address to the DKIM-signed message and resends it to a victim. When the victim's email server receives the message, its DKIM component may verify the original From header, and the message passes both DKIM and DMARC verification, while the MUA may show the fake From header. The attacker can induce such inconsistencies between DKIM components and email clients by exploiting the techniques described in Section 6.1 and Section 6.2.

7 Responsible Disclosure

We have reported all the vulnerabilities we discovered to both the affected vendors and to CERT/CC, and have received positive feedback from all vendors except Microsoft and Yahoo. Below we summarize their responses.

Gmail.com: fixed the A_3 and A_{18} attacks immediately after our report, and rewarded us with cash payments for the two attacks separately. They were investigating other attacks in our report.

Zoho.com: confirmed our report and have modified their servers to mitigate these attacks. They informed us that they already place some emails that potentially trigger the disclosed vulnerabilities into the receiving email users' "spam" folder, and that they monitor delivery metrics to determine whether to later reject them outright. They gave us four rewards, corresponding to the intra-server attacks, A_{16} attack, A_{18} attack and *UI-mismatch* attacks.

Protonmail.com: rewarded us for the intra-server attacks. They were looking at other attacks in the paper.

Mail.ru: rewarded us for the A_{18} attack and engaged in in-depth discussions with us about the specifics. For *UI-mismatch* attacks, they suggested the defense of MUAs warning users of possible spoofing attempts without affecting email delivery. They already provide authentication information to MUAs via Authentication-Results (RFC 8601) headers. As third-party MUAs are out of their control, they currently don't address spoofing attacks in third-party MUA interfaces. In the future, they would consider blocking emails with ambiguous addresses, but currently due not view doing so as feasible, since they observe too many cases of actual, valid messages with unusual headers.

Fastmail.com: told us that they generally don't consider email spoofing bugs for bug bounty purposes, but as our report provided a more notable finding than most, they offered us a cash reward in thanks.

Naver.com: confirmed our report and offered to include us as special contributors.

eM Client: discussed the attacks and possible defense solutions with us. They suggested that using a future IMAP extension, instead of the Authentication-Results header, could provie a more reliable way for email providers to report authentication information to MUAs. They stated they were assessing how to mitigate the issues we reported.

iCloud.com, Tutanota.com and Thunderbird: thanked us for our report and stated they were actively fixing these issues.

Microsoft: disregarded our report (which included our paper and a video [5] demoing the A_{10} attack) because the threats rely on social engineering, which they view as outside the scope of security vulnerabilities.

Yahoo.com: misunderstood our report (which included our paper and a video [6] demoing the attack in Figure 8b) as reflecting DNS misconfiguration issues, which we have clarified, but to date have received no further reply.

8 Discussion

The attacks we found share the high-level theme of inconsistencies between software components. We summarize three sources of inconsistencies that manifest in the overall picture.

First, the email protocols define multiple sender identifiers, leaving room for misaligned interpretations in implementations. For example, HELO and MAIL FROM commands, along with From, Sender, and Resent-From headers, represent different sending roles with similar or redundant semantics. While a strict specification can clarify and regulate protocol fields with confusing semantics, problems often arise when implementations lack a comprehensive understanding of the specifications.

Second, text-based protocols with complex syntaxes can lead to a variety of parsing inconsistencies. For example, the From header defines various complex features, for which different implementations can choose to implement different subsets. In addition, text-based protocols introduce flexible formatting and tolerance (e.g., allowing whitespace and comments to be freely inserted in many places), creating ample room for inconsistencies, especially when implementations vary in how they tolerate non-compliant inputs.

Finally, the process of sender authentication involves a chain of components, creating strong dependencies on implementation consistency and correctness. As shown in Figure 4, an email sent by the sender's MUA might be processed by at least six different components before reaching the recip-

[5] https://youtu.be/IsWgAEbPaK0
[6] https://youtu.be/DRepfStOruE

ient. Inconsistencies between any two components in the processing chain may introduce ambiguities.

All together, these elements create a tangled situation that human implementors and operators are unlikely to get it right.

8.1 Mitigation

We suggest a number of possible mitigations for these problems, ranging from immediate (mostly) implementation-level improvements, to broader considerations when designing protocols:

Following operational guide on DKIM specification to prevent replay attacks. RFC 6376 suggests that DKIM signers should include all important headers in DKIM signatures and avoid using the "`l=`" tag to prevent spoofing attacks.

RFC 6376 also suggests that DKIM signers should "oversign", i.e., repeat important headers, to prevent replay attacks, such as using "`h=from:from:subject:subject:to:to...`". This technique takes advantage of two DKIM features. First, each parameter of the "`h=`" tag matches a single occurrence of a header. Therefore, if a message has two Subject headers (which normally it will not), "`h=subject:subject`" will prevent an attacker from tampering with either of them. Second, DKIM allows signing nonexistent headers. For example, if a message lacks a Subject header, "`h=subject`" will prevent an attacker from adding one to the signed message.

Combining these two features, a domain owner can prevent replay attacks by setting "`h=from:from`" for messages with one From field. The first parameter signs the contents of the From header, and the second parameter guarantees that there is no additional From header. Any attempt to add an extra From header will break the signature. Among the 10 email providers we tested, only Yahoo.com adopts this solution. When we reported the vulnerabilities to Mail.ru, they informed us that they disabled this solution because of DKIM compatibility issues. However, they stated that they plan to re-enable it in Q1 2020.

Improving MUA display. MUAs would benefit from incorporating systematic consideration of how to better display security features. Most of the MUAs we tested do not display SPF, DKIM, or DMARC authentication results explicitly, making it difficult for end users, especially mobile client users, to apprehend the authentication status of the message. This lack facilitates attackers in bypassing server-side authentication, for example, by appending invisible characters to trick email servers into failing to obtain policy information via DNS. One possible approach for mitigating such attacks would be to add icons indicating emails with verified sender domains. We note however that experiences with such approaches for promoting HTTPS (via browsers displaying trusted icons for websites with valid TLS certificates) have demonstrated the challenges of ensuring that users correctly interpret the icons and do not get fooled by imposters [11–13].

We frame the above mitigations as "tactical": steps doable without significantly redesigning components or protocols. We now frame more strategic—but also more involved—mitigations.

Use of normalization. To defend against attackers using accounts on email services, email providers can consistently reset message headers (such as From) to remove potential ambiguities. However, the effectiveness of this approach still relies on correct parsing and interpretation of email MIME structures. We also caution that hardening a weak authentication system by composing it with additional security components, such as sanitizers or monitors, itself can introduce complex compositions that create new vulnerabilities, as we showed in Figures 6c and 6e.

Leveraging type systems to prevent internal inconsistencies. Some of our *intra-server* attacks, such as injection attacks, stem from inconsistent interpretations of messages between different internal components. Although implementors can address the specific attacks by filtering special characters that induce confusion, constructing fully correct filters can prove challenging. A more powerful implementation approach is to leverage a type system, such as using types to distinguish whether a field holds data or control information. If message forwarding between different components within a process preserves the type information, then injection threats can be addressed in a general fashion. However, employing this technique across disparate processes is more difficult, because for many communication frameworks the serialization and deserialization of messages (e.g., using JSON) can lose the necessary semantic information.

Avoiding re-processing. The root cause of *UI-mismatch* attacks is inconsistencies between email providers and MUAs. One possible mitigation solution[7] is for mail servers to provide authentication information to email clients *directly*, so that email clients can avoid re-parsing and re-verifying complex messages. Although RFC 8601 defines Authentication-results header to convey this information, the header itself can be forged by attackers. A more trustworthy way is to develop a future IMAP/POP3 extension that exposes the authentication results. The mail servers can pass the authentication information, including the verified address and verification results, to MUAs via IMAP/POP3 commands. The MUAs can then display the raw information exposed by mail servers without any additional parsing and verification.

Testing. To aid the community in securing additional email systems, we will make our testing tool publicly available via GitHub[8] after our reported issues are fixed by the vendors.

8.2 Discussion

That we could find so many attacks for widely used email services against their email authentication and integrity checks—crucial defenses against phishing and spear-phishing attacks—

[7] This idea comes from our discussion with eM Client.

[8] https://github.com/chenjj/espoofer

provides a wakeup call regarding the potential fragility of multi-component Internet services. While the specifics of the attacks reflect the particulars of various email protocols and mechanisms, in abstract terms the attacks leverage several classes of vulnerabilities likely present in other complex multi-component services.

In general, it is difficult to make components built by different developers fully consistent: 1) specifications allow for latitude in interpreting details; 2) it is easy to overlook the possibility of deliberate ambiguities in attacker-provided inputs; 3) specifications themselves evolve over time, with some components keeping outdated functionality for compatibility; 4) components can differ in which subset of a suite of complex features they implement; 5) components can vary in how they tolerate non-compliant inputs; and 6) functional equivalence-checking between complex components is intractable.

Many of the vulnerabilities we found arise not from programming mistakes but intended features. These features appear harmless when a component runs independently, but when integrated into a larger system, they introduce security issues. These attacks underscore a broad threat in modern system construction. Furthermore, the more complex a system's compositions, the more inconsistencies it may have, likely creating more vulnerabilities.

9 Related Work

Prior work discusses malformed email messages bypassing DMARC and DKIM [4, 14, 15]. Mailsploit encoded special characters such as newlines in From headers using an encoding approach given in RFC 1342 [14]. The author found that many email clients failed to properly sanitize such characters after decoding, leading to email-spoofing and code-injection attacks. This attack is similar but not the same as our A_{10} attack that uses encoding: his attacks encode control characters in From headers to exploit parsing errors in email clients, while our attacks encode spoofed email addresses to exploit inconsistencies between email servers and clients.

Replay attacks are a known problem noted in the DKIM specification [4], which in § 8 warns DKIM users of attacks involving extra header fields and the "`l=`" tag. But many developers overlook these warnings, and Ullrich presented multiple concrete attacks to exploit such weaknesses [15]. Based on the previous work, we introduce a new threat model to enhance the replay attacks. The previous replay attacks can't spoof the email body in DKIM-signed messages unless the target sites are misconfigured with the l= tag. Our attacks provide a new way to achieve this by combining replay attackers and malicious users of legitimate email providers.

These two efforts provided valuable initial considerations of the problem of bypassing email sender authentication mechanisms, and noted some of the complexities in parsing email messages. We build on this work by distilling the general theme of sender identity confusion due to inconsistencies between different components. Employing this theme facilitated us then identifying sources of ambiguities, enabling us to perform in-depth analyses leading to the discovery of a wide range of new attacks.

The email parsing inconsistencies our *UI-mismatch* attacks exploit can also exist in other systems, such as web applications. A previous writeup by Alderson showed that email address parsing inconsistencies in web applications can be exploited to take over accounts [16]. A recent blog by Davison discusses the possibility of exploiting address parsing inconsistencies between web applications and third-party sending services (e.g., Amazon SES) to bypass web application validation logic [17].

Another potential attack involving third-party sending services is cross-user spoofing, e.g., an SES user attempts to spoof another SES user's domain. We tested four popular third-party sending services (Amazon SES, SendGrid, Mailgun, and SparkPost) and found that none of them adequately validate From headers in messages: some (SendGrid, Mailgun) allow arbitrary From headers; some (SES, SparkPost) can be bypassed using the techniques developed in Section 5. Fortunately, all of them validate MAIL FROM and DKIM-Signature domains strictly (by verifying domain ownership), which makes DMARC bypassing difficult. But such services should consider addressing this issue anyway, because previous studies have shown that DMARC deployment and enforcement is problematic in practice [18–20].

Many researchers have conducted measurement studies on the deployment of SPF, DKIM, and DMARC [18–20]. Their results indicate that the adoption and enforcement of these extensions needs improvement. The community is actively promoting these security mechanisms—for example, the U.S. Department of Homeland Security requires all Federal agencies to deploy strict DMARC policies [21]. Our study shows that even in strict-deployment environments, attackers can still bypass these mechanisms.

Prior work has developed various phishing detection methods based on features extracted from email content, such as keywords, URLs, and attachments [22–24]. Our work focuses on how email systems authenticate the incoming messages; our attacks do not aim to bypass email content filters.

Recently, new protocols have been developed to enhance spoofing detection, such as BIMI (Brand Indicators for Message Identification) [6] and ARC (Authenticated Received Chain) [7]. BIMI is built on DMARC, and allows domain owners to coordinate with MUAs to display brand-specific indicators for DMARC-authenticated messages. ARC is built on SPF, DKIM and DMARC, and aims to address the authentication failure problem caused by modifications of mail forwarders. ARC allows each mail forwarder to append their authentication assessment results to the forwarded message, so that the receiving servers can make informed decisions based on authentication results from earlier forwarders. Since both BIMI and ARC rely on the correctness of DMARC verification, they are not helpful in preventing most of our attacks.

OpenPGP and S/MIME are two other standards to provide end-to-end authenticity of messages by digital signatures. Researchers have found many email clients to be vulnerable to signature spoofing or plain-text exfiltration attacks [25, 26]. Some of their attacks craft malformed MIME messages to exploit inconsistencies between signature verifiers and email display components. These attacks underscore an issue also highlighted by our work, namely that shifting sender authentication from email servers to clients cannot prevent email spoofing if inconsistencies exist.

Bratus et al. and Sassaman et al. proposed a formal language theory (LANGSEC) [27, 28] that provides a unifying framework regarding the root cause underlying the majority of software security problems: the complexity of the input language used in many real-world applications exceeds theoretical decidability bounds. These works advocate that protocol designers should restrict languages to lower levels of the Chomsky hierarchy to reduce parsing bugs and inconsistency bugs. Our attacks confirm the general problem they sketched; for example, many *UI-mismatch* attacks we found have their roots in the complexity of the From header syntax.

In addition to SMTP, inconsistency problems also exist in other computer systems, such as IP packet processing [9, 29–33], HTTP and web systems [34–39], file processing [40–43] and abuse of other operating system resources [44]. Handley et al. proposed "normalization" to rewrite network traffic to eliminate ambiguities between NIDS and end-hosts [9]. Wang et al. used verification-condition checking to identify inconsistent logic flaws in web payment systems [35]. Hooimeijer et al. designed the BEK language to analyze differences in sanitizers of web applications and mitigate XSS by using SMT solvers [45]. Brumley et al. proposed detecting discrepancies between different implementations by converting execution traces into symbolic formulae and comparing them using SMT solvers [46]. Some researchers have used differential fuzz testing techniques to identify discrepancies across different types of applications, such as C compilers [47], Java virtual machines [48], and SSL/TLS implementations [49–51].

10 Summary

Software components are supposed to make software less fragile and more reliable. In practice, however, part of the fragility is merely shifted from the component artifacts to the connectors and the composition process. When the composition is unreliable, composed systems can prove vulnerable.

In this paper, we illustrate the security implications of this problem in the context of modern email services. We present three classes of practical attacks against email authentication systems and identify a wide variety of inconsistencies between different components across email servers and clients. We show that these inconsistencies can enable an attacker to bypass email authentication to impersonate any site, and even forge DKIM-signed emails with a legitimate domain's signature. All 10 email providers and 19 MUAs in our experimental testing proved vulnerable to multiple of the 18 attacks that we developed.

As our software systems become increasingly complex, the need for building them out of disparate independent components rises. It appears likely that, in addition to email systems, many other real-world applications suffer similar problems. We hope this work can inspire the community to work towards securing additional applications.

Acknowledgments

We would like to thank our shepherd Devdatta Akhawe and the anonymous reviewers for their insightful comments. We are grateful to Haixin Duan, Zhiyun Qian, Michael Carl Tschantz, and Sadia Afroz for valuable discussion. We also thank Vladimir Dubrovin from Mai.ru, Filip Navara from eM Client, and securiy teams from other vendors for their helpful feedback. This work was supported in part by the National Science Foundation via grant CNS-1237265, and by a gift from Google. Opinions expressed in this paper do not necessarily reflect those of the research sponsors.

References

[1] G. T. Heineman and W. T. Councill, "Component-Based Software Engineering: Putting the Pieces Together," *Addison-Wesley*, p. 5, 2001.

[2] J. Klensin, "Simple Mail Transfer Protocol," Internet Requests for Comments, RFC Editor, RFC 5321, October 2008, http://www.rfc-editor.org/rfc/rfc5321.txt.

[3] S. Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1," Internet Requests for Comments, RFC Editor, RFC 7208, April 2014, http://www.rfc-editor.org/rfc/rfc7208.txt.

[4] D. Crocker, T. Hansen, and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures," Internet Requests for Comments, RFC Editor, STD 76, September 2011, http://www.rfc-editor.org/rfc/rfc6376.txt.

[5] M. Kucherawy and E. Zwicky, "Domain-based Message Authentication, Reporting, and Conformance (DMARC)," Internet Requests for Comments, RFC Editor, RFC 7489, March 2015, http://www.rfc-editor.org/rfc/rfc7489.txt.

[6] S. Blank, P. Goldstein, T. Loder, and T. Zink, "Brand Indicators for Message Identification (BIMI)," Working Draft, IETF Secretariat, Internet-Draft draft-blank-ietf-bimi-00, February 2019, http://www.ietf.org/internet-drafts/draft-blank-ietf-bimi-00.txt.

[7] K. Andersen, B. Long, S. Blank, and M. Kucherawy, "The Authenticated Received Chain (ARC) Protocol," Internet Requests for Comments, RFC Editor, RFC 8617, July 2019, http://www.rfc-editor.org/rfc/rfc8617.txt.

[8] Mozilla, "The public suffix list," https://publicsuffix.org/, 2019, [accessed Oct-2019].

[9] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," in *USENIX Security*, 2001.

[10] P. W. Resnick, "Internet Message Format," Internet Requests for Comments, RFC Editor, RFC 5322, October 2008, http://www.rfc-editor.org/rfc/rfc5322.txt.

[11] R. Dhamija, J. D. Tygar, and M. Hearst, "Why Phishing Works," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 581–590.

[12] C. Thompson, M. Shelton, E. Stark, M. Walker, E. Schechter, and A. P. Felt, "The Web's Identity Crisis: Understanding the Effectiveness of Website Identity Indicators," in *28th USENIX Security Symposium*, 2019, pp. 1715–1732.

[13] A. P. Felt, R. W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. E. Acer, E. Morant, and S. Consolvo, "Rethinking connection security indicators," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016, pp. 1–14.

[14] S. Haddouche, "Mailsploit," https://www.mailsploit.com/index, 2017, [accessed Oct-2019].

[15] S. Ullrich, "Breaking DKIM—on Purpose and by Chance," https://noxxi.de/research/breaking-dkim-on-purpose-and-by-chance.html, 2018, [accessed Oct-2019].

[16] E. Alderson, "Tchap: The super (not) secure app of the French government," https://medium.com/@fs0c131y/tchap-the-super-not-secure-app-of-the-french-government-84b31517d144, 2019, [accessed Feb-2020].

[17] N. Davison, "Exploiting email address parsing with AWS SES," https://nathandavison.com/blog/exploiting-email-address-parsing-with-aws-ses, 2020, [accessed Feb-2020].

[18] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman, "Neither snow nor rain nor MITM...: An empirical analysis of email delivery security," in *Proceedings of the 2015 Internet Measurement Conference*. ACM, 2015, pp. 27–39.

[19] I. D. Foster, J. Larson, M. Masich, A. C. Snoeren, S. Savage, and K. Levchenko, "Security by any other name: On the effectiveness of provider based email security," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 450–464.

[20] H. Hu and G. Wang, "End-to-end Measurements of Email Spoofing Attacks," in *Proc. USENIX Security Symposium*, 2018, pp. 1095–1112.

[21] U. D. of Homeland Security, "Binding Operational Directive 18-01: Enhance Email and Web Security," https://cyber.dhs.gov/bod/18-01/, 2017, [accessed Oct-2019].

[22] G. Ho, A. Sharma, M. Javed, V. Paxson, and D. Wagner, "Detecting Credential Spearphishing in Enterprise Settings," in *Proc. USENIX Security Symposium*, 2017, pp. 469–485.

[23] G. Ho, A. Cidon, L. Gavish, M. Schweighauser, V. Paxson, S. Savage, G. M. Voelker, and D. Wagner, "Detecting and Characterizing Lateral Phishing at Scale," in *Proc. USENIX Security Symposium*, 2019, pp. 1273–1290.

[24] A. Cidon, L. Gavish, I. Bleier, N. Korshun, M. Schweighauser, and A. Tsitkin, "High Precision Detection of Business Email Compromise," in *Proc. USENIX Security Symposium*, 2019, pp. 1291–1307.

[25] D. Poddebniak, C. Dresen, J. Müller, F. Ising, S. Schinzel, S. Friedberger, J. Somorovsky, and J. Schwenk, "Efail: Breaking S/MIME and OpenPGP email encryption using exfiltration channels," in *Proc. USENIX Security Symposium*, 2018, pp. 549–566.

[26] J. Müller, M. Brinkmann, D. Poddebniak, H. Böck, S. Schinzel, J. Somorovsky, and J. Schwenk, ""Johnny, you are fired!"–Spoofing OpenPGP and S/MIME Signatures in Emails," in *Proc. USENIX Security Symposium*, 2019.

[27] S. Bratus, M. E. Locasto, M. L. Patterson, L. Sassaman, and A. Shubina, "Exploit programming: From buffer overflows to weird machines and theory of computation," *USENIX; login*, vol. 36, no. 6, 2011.

[28] L. Sassaman, M. L. Patterson, S. Bratus, and M. E. Locasto, "Security applications of formal language theory," *IEEE Systems Journal*, vol. 7, no. 3, pp. 489–500, 2013.

[29] T. H. Ptacek and T. N. Newsham, "Insertion, Evasion, and Denial of service: Eluding Network Intrusion Detection," DTIC Document, Tech. Rep., 1998.

[30] R. F. Puppy, "A Look at Whisker's Anti-IDS Tactics," *Online*, 12 1999.

[31] M. Vutukuru, H. Balakrishnan, and V. Paxson, "Efficient and Robust TCP Stream Normalization," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 96–110.

[32] E. Korhonen, "Advanced Evasion Techniques—Measuring the Threat Detection Capabilities of Up-to-Date Network Security Devices," *Master's Thesis*, 08 2012.

[33] O.-P. Niemi and A. Levomäki, "Evading Deep Inspection for Fun and Shell," *Black Hat USA*, 2013.

[34] J. Chen, J. Jiang, H. Duan, N. Weaver, T. Wan, and V. Paxson, "Host of Troubles: Multiple Host Ambiguities in HTTP implementations," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1516–1527.

[35] R. Wang, S. Chen, X. Wang, and S. Qadeer, "How to Shop for Free Online–Security Analysis of Cashier-as-a-Service Based Web Stores," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 465–480.

[36] C. Linhart, A. Klein, R. Heled, and S. Orrin, "HTTP Request Smuggling," *Computer Security Journal*, vol. 22, no. 1, p. 13, 2006.

[37] S. Ullrich, "HTTP Evader—Automate Firewall Evasion Tests," http://noxxi.de/research/http-evader.html, [accessed Apr-2019].

[38] I. Ristic, "Protocol-level Evasion of Web Application Firewalls," *Black Hat USA*, 2012.

[39] J. Chen, X. Zheng, H.-X. Duan, J. Liang, J. Jiang, K. Li, T. Wan, and V. Paxson, "Forwarding-Loop Attacks in Content Delivery Networks," in *NDSS*, 2016.

[40] S. Jana and V. Shmatikov, "Abusing File Processing in Malware Detectors for Fun and Profit," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 80–94.

[41] J. Oberheide, M. Bailey, and F. Jahanian, "PolyPack: an Automated Online Packing Service for Optimal Antivirus Evasion," in *Proceedings of the 3rd USENIX Conference on Offensive Technologies*. USENIX Association, 2009, pp. 9–9.

[42] S. Porst, "How to Really Obfuscate your PDF Malware," *RECON, July*, 2010.

[43] D. Kaminsky, M. L. Patterson, and L. Sassaman, "PKI Layer Cake: New Collision Attacks Against the Global X. 509 Infrastructure," in *International Conference on Financial Cryptography and Data Security*. Springer, 2010, pp. 289–303.

[44] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," in *ACM SIGPLAN Notices*, vol. 41. ACM, 2006, pp. 372–382.

[45] P. Hooimeijer, B. Livshits, D. Molnar, P. Saxena, and M. Veanes, "Fast and Precise Sanitizer Analysis with BEK," in *USENIX Security Symposium*, vol. 58, 2011.

[46] D. Brumley, J. Caballero, Z. Liang, J. Newsome, and D. Song, "Towards Automatic Discovery of Deviations in Binary Implementations with Applications to Error Detection and Fingerprint Generation," in *USENIX Security Symposium*, 2007, p. 15.

[47] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and Understanding Bugs in C Compilers," in *ACM SIGPLAN Notices*, vol. 46. ACM, 2011, pp. 283–294.

[48] Y. Chen, T. Su, C. Sun, Z. Su, and J. Zhao, "Coverage-directed Differential Testing of JVM Implementations," in *ACM SIGPLAN Notices*, vol. 51. ACM, 2016, pp. 85–99.

[49] Y. Chen and Z. Su, "Guided Differential Testing of Certificate Validation in SSL/TLS Implementations," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 793–804.

[50] T. Petsios, A. Tang, S. Stolfo, A. D. Keromytis, and S. Jana, "NEZHA: Efficient Domain-independent Differential Testing," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 615–632.

[51] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 114–129.