Composition Kills: A Case Study of Email Sender Authentication

Jianjun Chen, Vern Paxson, and Jian Jiang

Component-based software design 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 work we provide a case study of such composition issues in the context of email (SMTP) sender authentication. We present 18 attacks for widely used email services to bypass their sender authentication checks by misusing combinations of SPF, DKIM and DMARC, which are crucial defenses against email phishing and spear-phishing attacks. Leveraging these attack techniques, an attacker can impersonate arbitrary senders without breaking email authentication, and even forge DKIM-signed emails with a legitimate site's signature.

Email spoofing, commonly used in phishing attacks, poses a serious threat to both individuals and organizations. Over the past years, a number of attacks used email spoofing or phishing attacks to breach enterprise networks [5] or government officials' accounts [10]. To address this problem, modern email services and websites employ authentication protocols—SPF, DKIM, and DMARC—to prevent email forgery. These protocols authenticate different aspects of email delivery, such as the sender's IP address (SPF), content integrity (DKIM), and correctness of the domains used for these checks (DMARC).

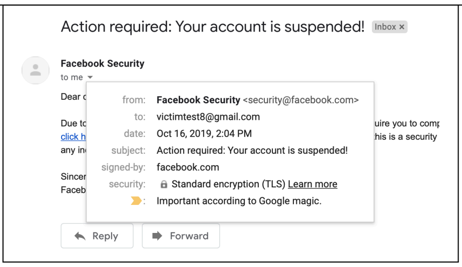
Our research in USENIX Security 2020 [3] identified a set of practical exploits to show the fragility of these protocols as implemented in practice. The key problem is that different components in the email processing chain employ a wide range of *inconsistent interpretation* regarding precisely how to interpret the different email elements they secure. Figure 1 illustrates one of our spoofing attacks to impersonating `facebook.com` by exploiting the inconsistency between the DKIM and DNS components. Gmail attests that the email was indeed from `security@facebook.com`, but in fact it was not: Gmail obtained the public key used for DKIM verification not from Facebook but instead from a server under our control (per Section *Case Study: Ambiguous-domains*). In total, we identified 18 types of similar attacks, all of which work by exploiting inconsistencies between different components across email servers and clients. We reported our findings to the affected vendors, who awarded us a number of bounties totalling several thousand dollars.

All of the attacks we found can be unified under a general theme—insecure composition, a rising threat in today's distributed systems. The techniques we developed can be applied to identify similar vulnerabilities in other systems. We published our testing tool on Github (`https://github.com/chenjj/espoofer`) to aid the community in securing additional email systems.

Background

SMTP lacks authentication

Simple Mail Transfer Protocol (SMTP) provides an Internet standard for mail transmission [9]. Figure 2 shows the three main steps to deliver an email message. When SMTP was originally designed, it had no built-in security mechanisms to authenticate the sender's identity. Therefore any Internet email user can impersonate another's identity by sending spoofed emails.



Action required: Your account is suspended! Inbox ×

Facebook Security
to me

Dear c

Due to
click h
any in

Sincer
Faceb

uire you to comp
his is a security

from: **Facebook Security** <security@facebook.com>
to: victimtest8@gmail.com
date: Oct 16, 2019, 2:04 PM
subject: Action required: Your account is suspended!
signed-by: facebook.com
security: Standard encryption (TLS) Learn more
: Important according to Google magic.

Figure 1: An example of our spoofing attacks to impersonates `facebook.com`. Gmail attests that this email is signed by `facebook.com`.

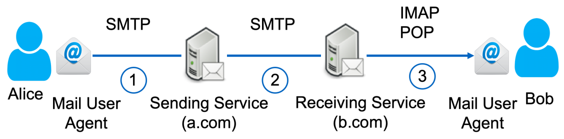


Figure 2: Email transmission from Alice to Bob

Which identity to authenticate?

The following example shows how an email is sent to a SMTP server via telnet:

```
HELO a.com
  250 - Mail b.com pleased to meet you
MAIL FROM: <sender@a.com>
  250 - Mail OK
RCPT TO: <receiver@b.com>
  250 - Mail OK
DATA
  354 - Enter mail, end with "." on a line by itself
From: Alice <alice@a.com>
To: <bob@b.com>
Subject: Hello from Alice

Hi Bob, I'm Alice.
.
  250 - Message accepted for delivery
```

SMTP's design includes multiple "identities" when handling messages. In the above example, both the MAIL FROM address and the From header address 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. For clarity of discussion, we will subsequently refer to the first as "MAIL FROM" and the second as "From".

In addition, SMTP introduces multiple other sender identities, such as in the HELO command, and the 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?

How SPF/DKIM/DMARC authenticate email senders

To combat email forgery, three email authentication mechanisms have been developed and widely deployed: SPF [8], DKIM [4], and DMARC [11].

SPF. Sender Policy Framework (SPF) uses the MAIL From and HELO address to authenticate the sender. When a message is received, the receiving mail server first queries the domain in the SMTP MAIL FROM and HELO commands to obtain the SPF policy, which contains lists of IP addresses authorized to send email for the domain. The receiving server checks if the sender's IP address matches the policy. If either HELO or MAIL FROM check fails, the mail server enforces the policy specified by the domain owner (e.g., hard fail, soft fail) to reject the message.

DKIM. DomainKeys Identified Mail (DKIM) authenticates senders using the `d=` field in a DKIM-signature header. When receiving a message, the receiving mail server first queries the domain in the `d=` field of

DKIM-Signature header to obtain the public key, and then verify the signed message with the obtained public key.

```
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. For our purposes, the relevant tags are:

- `d` represents the signer's *domain*.
- `s`, the *selector*, enables specifying multiple keys under the "`d=`" domain for fine-grained signatory control. The tag is used to obtain a public key by querying "s._domainkey.d" ("`selector._domainkey.example.com`" here).

Unfortunately, neither SPF nor DKIM provides a complete solution for preventing email spoofing, because 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.

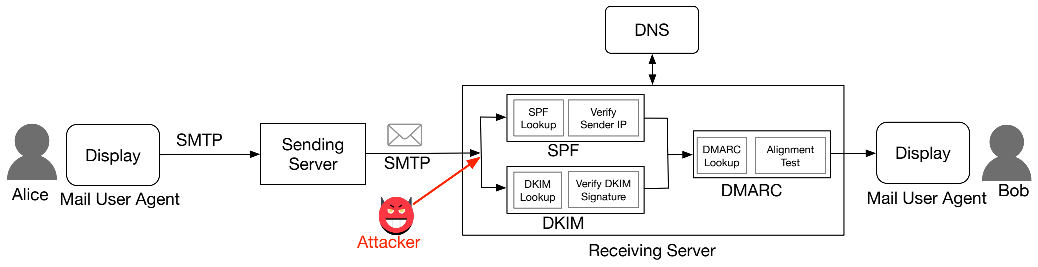


Figure 3: Email authentication flow. The attacker sends an email with a spoofed From header (`admin@legitimate.com`) to the victim directly from their mail server (`attack.com`), which—nominally—email authentication mechanisms should prevent.

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 needs to have the same registered domain (defined in the public suffix list, https://publicsuffix.org/). 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 is supposed to enable an email system to ensure that the address in the From header cannot be forged.

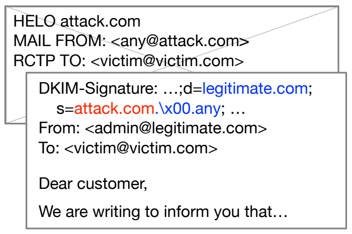
Our Discovery: Bypassing the Authentication

We found 18 different attacks that can bypass sender authentication. The key concept of our attacks is to exploit inconsistencies between different components in the email processing chain. As shown in Figure 3, an email sent by Alice might be processed by at least six different components before reaching Bob. Those components are often built by different developers or companies and can have a wide range of inconsistencies that attackers can exploit. In this article, we describe several representative case studies to illustrate how those inconsistencies could be exploited. More attacks can be found in our USENIX Security paper [3].

Case Study: Ambiguous domains

This example exploits inconsistencies between the DKIM component and the DNS component. An attacker can craft ambiguous domains to make the DKIM component believe that it's verifying the legitimate domain, but the

DNS component actually queries the attacker's domain to obtain those policy records. The DKIM component generates "pass" authentication results because the attacker controls the policy retrieved via DNS.


HELO attack.com
MAIL FROM: <any@attack.com>
RCTP TO: <victim@victim.com>

DKIM-Signature: ...;d=legitimate.com;
s=attack.com.\x00.any; ...
From: <admin@legitimate.com>
To: <victim@victim.com>

Dear customer,

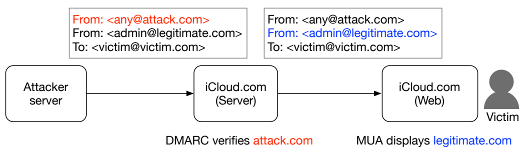
We are writing to inform you that...

Figure 4: NUL ambiguity. The DKIM component believes that the message is signed by `legitimate.com`, while the DNS component uses `attack.com` to obtain the public key.

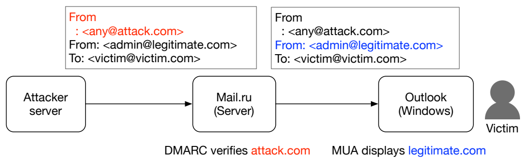
One way to craft such domains uses the NUL ("\x00") character, which terminates strings in some languages (e.g., C), but not in others (e.g., Perl or PHP). For example, we can fool Gmail.com using this technique. Gmail's DKIM and DNS components differ in interpreting NULs in the domain name, which we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

Here is how the attack works. Per Figure 4, the attacker constructs a fake email with arbitrary 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`".

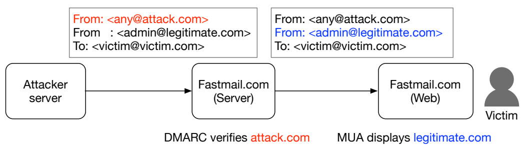
When the Gmail server receives the email, its DKIM component queries the domain *s._domainkey.d*, i.e., "`attack.com.\x00.any._domainkey.legitimate.com`", to obtain the public key. But when it resolves this domain, the DNS component parses the NUL character as a string terminator and instead obtains 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 matches the From header domain.



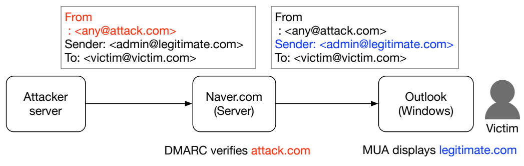
(a) Preference of multiple From headers.



(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.

Figure 5: Different cases of inconsistent interpretation of ambiguous From headers between email servers and MUAs.

Case Study: Ambiguous From headers

The second example exploits inconsistencies between mail servers and mail user agents (MUA) to make mail servers authenticate one sender identity, but MUAs display another sender identity to the end-user. The complexity and flexibility of email message syntax enabled us to find a large number of inconsistencies.

1) Multiple From headers. The simplest attack technique is to construct an email message with two different From headers. Figure 5a shows such an example. iCloud (Server) uses the first From header for DMARC verification and generates "pass" authentication results, but iCloud (Web) displays the second one (which is unverified) to the end-user.

Although RFC 5322 suggests that email messages with multiple From headers are invalid and should be rejected by receivers, 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; the other 11 MUAs display the first From header.

2) Space-surrounded From headers. We can further create different variants by inserting space around the From header. 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.

Whitespace abuse opens up new opportunities for multiple From ambiguities. First, the 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 this with a folding-space-succeeded From header, as shown in Figure 5b. Second, inconsistent interpretations 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 using the *same* header parsing and processing, by leveraging special forwarding behaviors of the email servers. Figure 5c 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. We can also bypass sender authentication by using From alternative headers. 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 like Sender or Resent-From header 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 a From header is not found, they don't perform DMARC authentication, or generate "none" as the result.

The interplay between From header and its alternative headers introduces another source of ambiguity. As shown in Figure 5d, 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.

Case Study: Ambiguous email addresses

Even if a mail server and client extract the same From header from an email message, extracting a *consistent* email address from that From header poses another challenge due to the complex syntax of From headers.

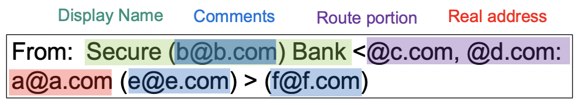
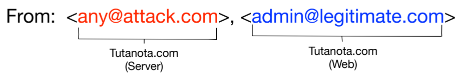

Display Name
Comments
Route portion
Real address
From: Secure (b@b.com) Bank <@c.com, @d.com:
a@a.com (e@e.com) > (f@f.com)

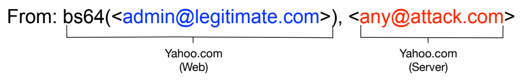
Figure 6: An example of a valid From header.

Figure 6 shows a valid From header with a single mailbox address, which consists of four elements.

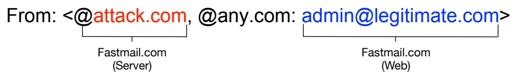
- *Display name* is an optional field that identifies the sender's name.
- *Real address* represents the real sender and is protected by SPF/DKIM/DMARC. (In this article, we aim to spoof the real address.)



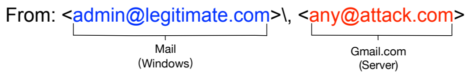
(a) Preference of multiple email addresses.



(b) Differences in parsing Base64-encoded address.



(c) Inconsistencies in supporting route portion feature.



(d) Differences in supporting quoted-pair feature.

Figure 7: Different cases of inconsistent interpretations of email addresses between email servers and MUAs.

- *Route portion* is an obsolete feature originally defined in RFC 822 to indicate the delivery path that the message should follow. RFC 5322 prohibits *generating* this obsolete field, but recipients still must *accept* it (and ignore the routing part).
- *Comments* provide some human-readable information and can be freely inserted in many places in 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. So the address in Figure 6 can be repeated multiple times, separated by a colon.
- *Encoding.* RFC 2047 defines encoding approaches to support non-ASCII characters in email headers. Its syntax looks like: `=?charset?encoding?encoded-text?=`, in which the "charset" field specifies the character set of the unencoded text; "encoding," which should be "B" or "Q", specifies the encoding algorithm; and "encoded-text," 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.
- *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.

Attacks leveraging complex From headers. We find that implementations vary in parsing and interpreting From headers. Here we show four sample attacks that exploit these inconsistencies, per Figure 7.

1) Multiple email addresses. 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 a new kind of ambiguity. The mail server may recognize a From header value that differs from the email address that the client displays. As shown in Figure 7a, Tutanota.com (Server) only uses the first address for DMARC checking, while its web interface only shows the second one.

2) Email address encoding. Figure 7b 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. As shown in Figure 7c, 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. Figure 7d 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.

Case Study: Forging DKIM-signed emails

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

First, we find that most email providers do not perform sufficient checks on the From header for mail originating from their local MUAs, enabling an attacker who has an email service account to send messages with fake From headers through the email provider's server. Since the exploited email providers will automatically attach DKIM signatures to their outgoing emails, such messages will pass the receiver's DKIM and DMARC validation.

Second, a *replay attacker* can bypass the check by adding an extra forged From header to a valid, previously sent, DKIM-signed message, and then *resend* the message to a victim. When the victim's email server receives the message, its DKIM component may verify the original From header, enabling the message to pass both DKIM and DMARC verification, while the MUA may instead show the *fake* From header. (For more details, see Section 6 of [3].)

Mitigation

We provide several suggestions for end-users, email clients, and email providers to mitigate these attacks.

End-users should not blindly trust the email sender displayed by email clients, since the sender address can be spoofed. Users should carefully examine or verify the message content, especially when the message impels the user to take some action.

Email clients should have a 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.

Email providers. The root cause of our attacks lies in the inconsistent interpretation of ambiguous messages. One possible mitigation is that email providers parse email messages strictly, and block messages with ambiguous addresses. To aid the community in identifying such security issues, we have released our testing tool on GitHub [1] to help administrators and security practitioners secure email systems.

Conclusion

We presented a set of practical attacks against email authentication systems and identified a wide variety of inconsistencies between different components across email servers and clients. We showed 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 that, in addition to email systems, many other real-world applications suffer similar problems [2, 6, 7]. We hope this research can inspire the community to work towards developing methodologies to help secure additional applications.

References

[1] Espoofer. https://github.com/chenjj/espoofer, 2020. [accessed Dec-2020].

[2] Jianjun Chen, Jian Jiang, Haixin Duan, Nicholas Weaver, Tao Wan, and Vern Paxson. Host of Troubles: Multiple Host Ambiguities in HTTP implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1516–1527. ACM, 2016.

[3] Jianjun Chen, Vern Paxson, and Jian Jiang. Composition kills: A case study of email sender authentication. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

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

[5] Sean Gallagher and David Kravets. How did yahoo get breached? employee got spear phished, fbi suggests. https://arstechnica.com/tech-policy/2017/03/fbi-hints-that-hack-of-semi-privileged-yahoo-employee-led-to-massive-breach/, 2017. [accessed Dec-2020].

[6] Mark Handley, Vern Paxson, and Christian Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *USENIX Security*, 2001.

[7] James Kettle. HTTP Desync Attacks: Smashing into the Cell Next Door. *Black Hat USA*, 2019.

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

[9] J. Klensin. Simple Mail Transfer Protocol. RFC 5321, RFC Editor, October 2008. http://www.rfc-editor.org/rfc/rfc5321.txt.

[10] Katiana Krawchenko. The phishing email that hacked the account of john podesta. https://www.cbsnews.com/news/the-phishing-email-that-hacked-the-account-of-john-podesta/, 2016. [accessed Dec-2020].

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