Email Spoofing with SMTP Smuggling

Ryan Daly
ryan4@vt.edu
Virginia Tech
Blacksburg, USA

Gayathri Mahendran
gayathrimahen@vt.edu
Virginia Tech
Blacksburg, USA

Sutton Marks
smarks27@vt.edu
Virginia Tech
Blacksburg, USA

Abstract

This report investigates the susceptibility of modern email providers and software to SMTP smuggling, a recent vulnerability arising from inconsistencies in message parsing that enables an attacker to include an additional message which bypasses authentication checks. To study this, we examined the behavior of public email providers and open-source mail software when sending and receiving specially crafted messages. We then analyzed message delivery by checking email inboxes and raw network traffic to determine what mitigations, if any, had been implemented. We found that all the previously reported weaknesses in the systems we tested have been patched, confirming that SMTP smuggling is no longer impacting the most popular and up-to-date email systems. However, through a brief analysis of thousands of mail server banners, we determined that many devices are actively using vulnerable software versions, suggesting that many mail servers could still be susceptible to SMTP smuggling. Our results show that while public providers and open-source email software were quick to patch their systems, delays in updating to the latest software version may leave many mail systems vulnerable.

Keywords

SMTP Smuggling, email spoofing, Mail Transfer Agents (MTAs), email security, SPF/DKIM/DMARC bypass, parsing inconsistencies, message boundary parsing

1 Introduction

SMTP is one of the most widely deployed communication protocols on the Internet, yet much of its design predates modern security expectations. Over time, a series of extensions and independent Mail Transfer Agent (MTA) implementations have introduced subtle differences in how servers parse and interpret message boundaries. In 2023, researchers showed that these inconsistencies could be combined into a practical attack technique called SMTP smuggling. Later on at USENIX 2025, these researchers presented and published their study on these findings. In the paper titled "Email Spoofing with SMTP Smuggling: How the Shared Email Infrastructures Magnify this Vulnerability", Wang et al. [1] shows that by exploiting how certain MTAs handle newline characters and end of data markers, an attacker can embed an additional email inside a single SMTP session. The smuggled message bypasses authentication checks and appears to originate from a trusted sender, effectively enabling email spoofing even in systems that enforce SPF, DKIM and DMARC.

The original study demonstrated that several popular MTAs including Postfix, Exim and Sendmail were affected by variations of this parsing flaw. Major email providers quickly deployed patches once the vulnerability became public, but the broader ecosystem of self-hosted and open-source mail servers is far more diverse and slower to update. This raises a question: even if large providers have fixed the issue, how much exposure remains across the rest of the Internet?

In this project, we set out to replicate the core ideas from the SMTP smuggling paper and examine how modern MTAs behave today. Our work has three components. First, we tested several major public email providers to see whether any remained susceptible to crafted payloads resembling those used in the original attack. Second, we installed and configured widely used open-source MTAs to observe how current versions handle the same inputs. Finally, we conducted a lightweight banner analysis of thousands of publicly accessible SMTP servers to understand how many still show versions associated with the vulnerability.

Together, these steps allowed us to evaluate not only whether patched systems behave as expected, but also whether the ecosystem still contains a meaningful number of outdated servers that could theoretically be at risk. Rather than reproducing the full scale of the original experiment, our goal was to understand present-day exposure and see how quickly different segments of the email infrastructure responded to the disclosure of SMTP smuggling.

2 Methodology

Our methodology follows the structure of the original SMTP smuggling study, adapted to reflect modern deployments and practical testing constraints. Rather than attempting a full reproduction, our goals was to evaluate whether the core attack behaviors still appear in current systems and how modern MTAs handle malformed message boundaries.

We evaluated SMTP smuggling behavior across three settings: public email providers, locally deployed open-source MTAs and publicly visible SMTP servers via banner analysis. Each component captures a different stage of the email delivery pipeline and highlights potential inconsistencies in message parsing.

2.1 Public Provider Testing

Public provider testing consisted of both receiving side and sending side experiments. For receiving tests, we connected directly to provider mail servers and transmitted crafted SMTP DATA payloads designed to manipulate message terminators. Inbox delivery and message content were inspected to determine whether malformed payloads were normalized or rejected.

For sending tests, we authenticated to public email services and sent crafted payloads to a controlled receiving MTA. Network traffic was captured on port 25 to observe whether payloads were modified during transmission. Payloads consisted of newline and delimiter variations previously shown to enable SMTP smuggling.

CS5584 Fall '25, Blacksburg, VA
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2 Open-Source MTA Testing

Open source testing was conducted using Postfix and Exim deployed in a controlled local environment. Two physical Windows 11 machines communicated over a local network with Linux based tooling provided via WSL2(Ubuntu). SMTP communication occurred over IPv4 on port 25.

Crafted payloads were transmitted using command line utilities such as mailutils and packet level traffic was captured using tcpdump and Wireshark. Both sending and receiving side roles were evaluated to observe how each MTA handled malformed message boundaries under current configurations

2.3 Banner Analysis

The banner based analysis was done using publicly available SMTP banner data obtained from Shodan. SMTP banners disclose MTA software and version information during the initial handshake, offering insight into patch adoption trends.

Extracted MTA versions were compared against the vulnerability timelines reported in the original SMTP smuggling paper. While banner data cannot confirm exploitability, it provides a high-level view of how widely outdated MTA versions remain deployed.

3 Public Provider Testing

3.1 Motivation

Public email services such as Gmail and Outlook form the backbone of the global email infrastructure, providing clients with a straightforward and reliable way to send and receive email. Most people who use personal email accounts rely on public email providers every day, making the security of those systems paramount.

3.2 Infrastructure

To test current susceptibility to SMTP smuggling, we configured a testing environment capable of delivering messages to public email services. We focused on developing a reproducible setup that would allow testing regardless of local restrictions on port 25 connections. Furthermore, we prioritized the ability for all collaborators to connect and perform tests. To satisfy those system design goals, we opted for a cloud environment that could send and receive traffic on port 25. After finding that the most popular cloud providers such as Amazon Web Services and Google Cloud Platform restrict outbound port 25 activity, we identified a European provider, Webdock, that suited the needs of our environment.

Next, we created two virtual machines through that provider, and were allocated static IP addresses for them. One virtual machine was intended to perform receiving tests of the public email providers by sending them malicious messages. The other virtual machine would receive messages from the public providers, and we would analyze the contents as the data arrived. While the two systems could have been combined to reduce the complexity of the setup, we chose to separate them since they represented distinct functionality.

To ensure that the messages were successfully delivered, we registered a domain name and configured the DNS records accordingly. First, we set the A record to match the static IP address of the VM which delivers mail to the public providers, and the MX record to match the static IP address of the VM that receives mail from them. Then, we added an SPF TXT record authorizing the first VM to send email on behalf of the domain, as well as a DMARC TXT record after registering with Dmarcian. Finally, we created a public-private key pair and published the public key in a DKIM TXT record. This configuration ensured that the necessary machines in our environment could successfully deliver messages to public providers without failing authentication checks and other email security checks.

3.3 Receiving Tests

We began our analysis by examining how public providers interpreted messages containing malicious payloads. We define a malicious payload to be a sequence of bytes other than the intended \r\n.\r\n used to separate a legitimate SMTP DATA section from a new message. In accordance with the methodology of Wang et. al [1], we used the following payloads:

\n.\n	\n.\r	\r.\n
\r.\r	\n.\r\n	\r.\r.\n
\r\n\x00.\r\n	\n\x00.\n	\n\x00.\r
\r\x00.\n	r\x00.\r	\n\x00.\r\n
\r\x00.\r\n		

While we began implementing our own test harness, we decided instead to use and adapt the script developed by Wang et. al [1] to prevent confounding variables that could compromise our comparison. By doing so, we guaranteed that any discrepancies we observed were due to changes in the mail providers and software as opposed to differences in our code.

We attempted to create email accounts for the public email services outlined in the original paper, and were successfully able to register and authenticate to 9 of them: Gmail, Yahoo, Zoho, AOL, Yandex, Mail.ru, Outlook, iCloud, and Daum. For several of the other providers, such as Aliyun and Naver, either a phone number in the service's home country was required, or we received alerts of suspicious login activity likely based on our location in the US. We investigated whether any alternatives existed and explored potentially using temporary phone number services to bypass the restriction. Unfortunately, most of those attempts were unsuccessful and we could only test the previously listed providers at this time.

Before performing the tests, we updated the test harness to sign the messages with the DKIM private key corresponding to the public key in the DNS record. Furthermore, we altered the sender information in the original paper's tests to match our domain so the message would pass SPF validation and be successfully delivered. We also supplied the Python test harness with the account addresses and additional parameters needed to connect such as whether STARTTLS was necessary.

We used that setup to deliver confirmation emails to each of the accounts we registered, allowing us to verify that the later tests could be received and that SPF, DKIM, and DMARC validation were successful. By checking the account inboxes, we confirmed that the messages were successfully delivered from the account on our domain and that our infrastructure was properly configured to support these tests. Next, we used the test harness to send a

Table 1: Observed Receiving-Side Behavior of Public Email Providers

Provider	Vulnerable (2023)	Vulnerable (2025)	Observed Behavior (2025)	Invalid DKIM
Google	No	No	Malformed terminators normalized or ignored	8/13
Outlook	No	No	Truncates 8/13 malformed payloads; differs from 2023 behavior	13/13
Daum	Yes	No	Previously vulnerable to 5 payloads, now mitigated	N/A
AOL	No	No	Malformed terminators normalized or ignored	9/13
Yahoo	No	No	Malformed terminators normalized or ignored	5/13
Zoho	No	No	Malformed terminators normalized or ignored	8/13

message containing each of the payloads to all of the accounts we set up. We monitored the SMTP conversations by enabling verbose logging features, allowing us to verify that messages were accepted by the public providers as they were sent, and to diagnose any deliverability issues. We then inspected the message inboxes of the public email providers to determine if SMTP smuggling was successful. If so, the message embedded within the original following the malicious payload would appear as its own message in the inbox.

Since public providers do not provide a way to examine the raw bytes of the messages as they were received and stored, we relied on another mechanism to assess whether payloads were being canonicalized or ignored. In particular, we examined the DKIM validation status of the received messages, focusing on changes to the body hash. All of the providers we examined allowed inspection of the original message source except for Daum, though fewer provided details about the cause of DKIM validation failures. Because the confirmation emails were successfully validated and nothing changed in the experiment other than the payload used, we were able to attribute DKIM validation status changes to differences in how the messages were interpreted based on the payload they included.

However, depending on the DKIM canonicalization strictness used, some messages could have passed validation even after undergoing minor changes. That level of strictness, known as relaxed, could have enabled messages to pass validation even if the email providers normalized the payloads. For the purposes of our evaluation, we chose to use simple canonicalization, which causes validation to fail if minor changes are made to the message. This would enable us to identify which payloads were altered upon receipt by analyzing the DKIM validation status. As we could not view the raw bytes of the messages in the inbox, this technique provided needed insight into how aggressively various public providers are normalizing payloads and which are primarily ignoring malformed end-of-data terminators.

As the results in Table 1 show, the public providers we tested demonstrated similar behavior as found in the previous study, with one provider, Daum, no longer susceptible to any of the 5 payloads it had been susceptible to. Furthermore, we observed that only Outlook truncated messages at the improper terminator, while the remaining providers displayed the combined messages as one. Additionally, Outlook was found to use truncation for different payloads than those previously observed by Wang et. al [1].

In addition to assessing how public email providers have implemented or changed their defenses against SMTP smuggling since the work of Wang et. al [1], our work also analyzed which payloads were correlated with DKIM validation failures. We observed that all 13 payloads led to DKIM validation failures in Outlook due to changes in the body hash, indicating that Outlook may be strictly enforcing canonicalization rather than just ignoring payloads. We also found that the following payloads corresponded with fewer DKIM validation failures across providers compared to other payloads, with one or two failures compared to five. This suggests potential consistency in how public providers address certain payloads.

3.4 Sending Tests

While receiving-side vulnerabilities pose the greatest threat to public email providers when not mitigated, we also investigated whether those providers could be used to send smuggled SMTP messages. Since raw bytes cannot be entered in standard email clients and websites, we relied on Wang et al.'s [1] test harness to authenticate to the public providers' mail transfer agents and submit raw messages for delivery. Their work relies on the public providers supporting password-based authentication from external applications. While some providers still support this, others are transitioning to more secure authentication mechanisms such as OAuth2, which restricted the number of providers we were able to test.

As previously mentioned, the MX record of the domain we registered points to a cloud-based virtual machine, which we configured to receive messages delivered to addresses on the domain. We did so by installing Postfix, a popular open-source MTA, which the public providers can communicate with and deliver messages to. Next, we installed Tshark on the virtual machine to monitor incoming SMTP traffic. For a public provider to be considered vulnerable on the sending side, it must deliver messages containing invalid terminators as submitted by the testing harness. By analyzing the raw SMTP traffic on the receiving mail server, we were able to determine what changes were made to the payload by the public provider. We did not look at the inbox of Postfix, as Postfix itself could normalize malicious payloads delivered by a public provider, making it impossible to tell whether the public provider or open-source MTA was responsible for the normalization.

After enabling app passwords where possible for the 8 accounts we created, we gathered details about each provider's SMTP server for submissions. We then modified the test harness to apply the authors' code to submit messages to the providers we tested and deliver them to the receiving VM we managed. After attempting to send confirmation emails to the providers that supported app passwords, only three providers accepted messages into their queues. The other providers responded with authentication errors or generic

Table 2: Observed Sending Side Behavior of Public Email Providers

Provider	Vulnerable 2023	Vulnerable 2025	Observed Behavior 2025	Changes
Google	Yes	No	All payloads normalized	All 7 issues patched, no dot stuffing
iCloud	Yes	No	Malicious messages dropped	No longer delivering messages with payloads
Zoho	Yes	No	All payloads normalized, first 6 dot-stuffed	All 7 issues patched

error messages when submitting messages to them. In case the errors were due to the recent creation of the app password, or too many authentication attempts, we waited for extended periods before attempting again. However, only Gmail, Zoho, and iCloud reliably allowed authentication from the test harness and accepted messages into their queues. All three of those providers successfully delivered the confirmation message to the receiving virtual machine, based on our analysis of incoming traffic on port 25.

Using the test harness, we submitted one message for each of the 13 payloads across all three providers. We then recorded the incoming traffic on port 25 of the receiving virtual machine, resulting in an SMTP packet capture for each of the providers. Notably, we observed that no SMTP messages were delivered from iCloud despite successful delivery of the confirmation message that did not contain a malicious payload. To verify that the failed delivery was associated with the message content and was not a temporary failure, we resent both the confirmation email and test messages over the course of several hours. Across several trials, the confirmation email was successfully delivered while the messages containing malicious payloads were not.

While all three providers were vulnerable to at least one payload in the previous study published last year, our results show that they are no longer vulnerable using any of the 13 payloads. By analyzing the raw bytes of the packet captures using Wireshark,weWe observed consistent canonicalization in the traffic sent through Gmail and Zoho. The payloads not containing a null byte were normalized to become the standard CRLF.CRLF end-of-data terminator, while those containing a null byte became the standard terminator with a null byte in the middle. Zoho employed dot stuffing, or the insertion of an extra period in the end-of-data indicator, while Gmail did not. This represented a change from the findings of Wang et al. [1], who found that Gmail consistently employed dot stuffing. While testing additional providers would provide stronger insight into the overall state of sending-side vulnerabilities, our results as shown in Table X reveal that some of the most popular providers have improved their mitigations through aggressive normalization and potentially dropping malformed messages.

4 Open Source Testing

4.1 Environment

The motivation of the open source MTA testing is to individually examine how various popular MTAs react to and treat SMTP smuggling. SMTP messages travel over a chain of MTAs in the wild. If any one of the links in said chain break, then the whole chain could in theory be compromised. Therefore, it is important that we test publicly available MTAs, but also popular open source agents that appear throught the Internet.

Two of the most popular open source MTAs are `postfix` and `exim`. These two MTAs are widely deployed all over the Internet. Therefore, in our methodology we have chosen to evaluate `postfix` and `exim`, however, this research can extended to any other open source MTAs.

The experiment for the environment uses two physical machines, both running Windows 11 for host operating system. On both machines, WSL (Windows Subsystem for Linux) is used to virtually layer Linux on top of Windows, which is key for the network configuration. Furthermore, On both machines, `postfix` 3.10.5 and `exim` 4.9.7 are installed, as well as `mailutils` for a mailbox. The network type used for experimentation is a 802.11 WLAN. This requires routing for both operating system layers needs to be configured properly so that both machines can communicate on port 25(SMTP). This is done via `netsh` in Windows, `netcat` in WSL(Linux), and the individual MTA config settings. `socat` with `echo` is used to transmit messages from one machine onto the network over port 25, with `stty` also being used to avoid any extra delimiters being added to the emails.

Once properly configured, the two machines are able to transmit SMTP emails to one another, but we can also capture SMTP messages between machines. For evaluating only MTA receivers, `netcat` is used to transmit directly over port 25 to the receiver MTA, since our sender MTAs often caught smuggle attempts first. Further, `tcpdump` is used capture packets and `wireshark` is used to analyze the packets. With this lab environment and the analysis tools together, we can evaluate the viability of email smuggling via SMTP spoofing over a variety of CRLF patterns. This environment is extremely important because with our methodology, the receiver inbox will always receive the smuggle as two emails. However, when we can capture the data packets between two MTAs we can see if a smuggle attempt is accepted as one or two emails i.e. whether the attack succeeds or not.

4.2 Evaluation

For analysis of the open source MTAs, we chose only 8 CRLF patterns to evaluate due to time constraints. Specifically, these payload patterns are as follows:

```
\n.\r\n , \n\n.\r\n , \r\r.\r\n , \r\n\r.\r\n ,
\r.\n , \n.\r , \r.\r\n , \n\r.\r\n
```

Note that Wang et al. [1] examined more patterns, specifically those with 0 bytes. However, we chose not to examine these patterns due to time constraints. See the Extensions and Next Steps section below. Packets were captured on both the outgoing side of the sender MTA, and the incoming side of the receiver MTA in order to evaluate where packet changes were being made. Both exim and postfix were tested on the sender and receiver MTA sides for

all 8 CRLF patterns. Furthermore, during our literature analysis, we could not find any existing papers evaluating the differences `ipv4` versus `ipv6` have on SMTP smuggling. Therefore, we used our environment (specifically `netsh` and `netcat`) to change IP routing configurations and evaluate MTAs over both `ipv4` and `ipv6`. We quantify, a CRLF payload pattern as rejected if the single SMTP email is split into two separate SMTP emails, meaning the smuggle attempt was rejected. On the other hand, if the smuggling SMTP email successfully spoofs the second email then we quantify the CRLF pattern used as accepted.

4.3 Results

As shown in Table 3, none of the 8 CRLF payload patterns tested appeared successful. Both postfix and exim caught the payload anomaly and adjusted the SMTP packets appropriately for both the sender and receiver sides. This is likely due to a combination of dot stuffing and string filtering. However, this is in line with the results from the paper by Wang et al. [1], where `A7-A13` patterns contain 0 bytes. This is why it is key for us to examine patterns with 0 bytes next, but also the same open source MTA versions as Wang et al. [1] because many in-the-wild MTAs likely use these and older versions. Furthermore, as stated in above in Evaluation, we examined the differences `ipv4` and `ipv6` routing can have on this type of SMTP smuggling. Table 3 shows that there is no discernible difference Internet routing has on the tested payload patterns, but other patterns and MTAs need to be further tested.

5 Banner Analysis

To understand whether SMTP smuggling vulnerabilities may still persist in the broader ecosystem, we conducted a banner-based analysis of publicly visible mail servers. Banner information is typically returned during the initial SMTP handshake and often includes the software type and version number of the MTA. Because the original paper showed only the specifics of widely deployed MTAs were susceptible to smuggling, identifying which versions in active use provides insight into potential real-world exposure.

For this analysis, we examined 3,000 mail server banners, with 1,000 each for Exim, Haraka, and Sendmail. This data was collected from Shodan to abide by ethical data collection practices since we did not seek permission for a broader scan. We were constrained by academic tier limits, but sought to use passive data sources to gain insight into software versions in use. The dataset includes the banner information returned each time a connection is made on port 25, as well as device-specific data such as IP address and domain name when available. Previous analyses of banner data by Wang et al. [1] focused on shared mail infrastructure of top domains, while our approach aims to evaluate independently-hosted open source mail software. Banner data alone cannot confirm exploitability, but it provides a useful snapshot of software deployment trends and patch adoption.

Across the dataset, we extracted each banner's reported MTA type and version and identified unique MTA/version combinations. We then compared these versions against the vulnerability thresholds presented in the paper. Systems running versions older than patches released in late 2023 were classified as potentially vulnerable, while newer versions were classified as patched. We observed the following percentages of potentially vulnerable software:

Provider	% Potentially Vulnerable
Exim	22.0
Haraka	30.7
Sendmail	95.0

Table 4: Potentially Vulnerable Software Percentages

This analysis revealed that although the major public email providers have fully deployed patched versions of their MTA, as confirmed in our public testing, older versions of Exim, Postfix and Sendmail remain widely deployed in the open Internet. While a larger sample size would have allowed for higher confidence regarding the global state of open source email software versions, our results show that many mail servers are still using outdated software that could be susceptible to SMTP smuggling. Because many banners correspond to versions implicated in inconsistent newline parsing, these servers may remain susceptible in theory. Although banner data cannot confirm configuration details, it highlights that a meaningful portion of MTAs still run outdated software.

Overall, the banner analysis supports our main findings: major providers patched quickly, but many smaller or self-managed servers remain outdated, indicating that SMTP smuggling may still be risky in environments with slow update cycles.

6 Limitations

While our study provides insights into how modern MTAs handle SMTP smuggling related payloads, several limitations shaped the scope of our results.

First, our testing environment relied on the behavior of publicly accessible email providers and open-source MTAs installed locally. This allowed us to reproduce the core ideas from the original smuggling paper but did not capture the full diversity of real-world mail configurations. Many MTAs deploy custom policies, specialized filtering rules that we could not fully replicate in our setup.

Second, our open-source testing was limited to two of the most widely used MTA families - exim and postfix, and only specific versions available through current distributions. Older or very customized deployments may behave differently. In particular, patches released after SMTP smuggling was disclosed have been adopted inconsistently across the ecosystem, meaning our local tests may not reflect how unpatched systems behave today. This demonstrates the need for using this environment and methodology to test older versions and more open source MTAs.

7 Proposed Extensions and Mitigations

7.1 Mitigations

Our results suggest that most major providers and the latest open-source MTA releases have adopted defensive parsing behaviors that mitigate SMTP smuggling attacks. These defenses include stricter newline normalization, correct dot stuffing and filtering of malformed message terminators. As a result, the specific attack

Table 3: Observed Open Source MTA CRLF Payload Pattern Acceptance over Network Layer

IPv4	`\n.\r\n`	`\n\n.\r\n`	`\r\r.\r\n`	`\r\n\r.\r\n`	`\r.\n`	`\n.\r`	`\r.\r\n`	`\n.\r\r\n`
`postfix` (sender)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
`exim` (sender)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
`postfix` (receiver)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
`exim` (receiver)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
IPv6	`\n.\r\n`	`\n\n.\r\n`	`\r\r.\r\n`	`\r\n\r.\r\n`	`\r.\n`	`\n.\r`	`\r.\r\n`	`\n.\r\r\n`
`postfix` (sender)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
`exim` (sender)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
`postfix` (receiver)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
`exim` (receiver)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

Key: **X** = rejected, **O** = accepted

variants shown in prior work were not reproducible against up to date systems in our tests.

Based on the SMTP smuggling threat model, effective mitigations fall into two broad categories: implementation-level hardening and operational controls.

Implementation-level hardening

First, MTAs should strictly enforce the RFC compliant end of DATA sequence (`\r\n.\r\n`) and reject or normalize ambiguous newline representations before further processing. Inconsistent handling of newline characters across different processing stages is a core requirement for SMTP smuggling.

Second, correct and consistent dot-stuffing must be applied throughout the mail handling pipeline. Message boundary parsing should be identical across SMTP front-ends, filtering components and downstream processes such as DKIM signing or verification, ensuring that no additional message content can be interpreted after authentication checks.

Finally, MTAs should defensively sanitize unexpected or non standard bytes in boundary contexts. Payload families that exploit edge cases in byte level parsing have historically led to inconsistencies between components, making explicit rejection or normalization of such inputs a necessary safeguard.

Operational controls

From an operational perspective, timely patch adoption remains the most effective defense. The exploitability of SMTP smuggling is tightly coupled to specific MTA versions and parsing behaviors, making outdated deployments a primary risk factor.

7.2 Extensions and Next Steps

Although our testing suggests that commonly used providers and current open source MTA releases are no longer vulnerable to known SMTP smuggling techniques, several extensions would improve confidence in the results and better characterize residual risk in the ecosystem.

First, expanding the banner-based analysis using large scale Internet measurement data would allow for a more precise estimate of how many publicly reachable SMTP servers continue to advertise older or potentially vulnerable MTA versions. Such an analysis could also reveal patch adoption trends over time.

Second, testing a broader set of public email providers, including regional or less common services, would help address coverage gaps introduced by account creation or verification constraints encountered during this study.

Finally, extending evaluation to additional open source MTAs and older software versions would provide insight into downgrade risk and the long tail of legacy deployments. While current releases appear robust, long-lived or minimally maintained systems may still exhibit behaviors similar to those identified in the original SMTP smuggling disclosure.

The groundwork for our environment in evaluating email spoofing via SMTP smuggling has been laid. The major goals of our research has been to first replicate, evaluate, and validate the results and lab of Wang et al. [1] at a smaller scale. and then also add any novel extensions possible, such as evaluating how the networking layer affects SMTP smuggling. The next steps of our research is to continue to work on adding the extensions listed above. Current status of this open source project can be found on GitHub. Raw results, source code, and further documentation can be found there as well.

8 Conclusion

Our investigation into the modern state of SMTP smuggling defenses showed that all of the tested public providers have successfully deployed mitigations. Most rely on normalizing end-of-data terminators, and they often do not truncate messages. We proposed a method of using DKIM validation status to gain insight into how public email providers are transforming malicious messages during processing. Mainly, current versions of MTAs are typically patched for the smuggling attacks examined. However, older versions and less popular MTAs are still vulnerable. There is a clear need to further research the state of in-the-wild MTAs vulnerabilities to email spoofing via SMTP smuggling.

References

[1] Rui Wang, Yue Zhang, et al. 2024. Email Spoofing with SMTP Smuggling: How the Shared Email Infrastructures Magnify This Vulnerability. In *Proceedings of the 33rd USENIX Security Symposium*. USENIX Association, Philadelphia, PA, USA.