Inbox Invasion: Exploiting MIME Ambiguities to Evade Email Attachment Detectors

Jiahe Zhang
Tsinghua University
Beijing, China
zhjh23@mails.tsinghua.edu.cn

Jianjun Chen*
Tsinghua University; Zhongguancun Laboratory
Beijing, China
jianjun@tsinghua.edu.cn

Qi Wang
Tsinghua University
Beijing, China
qi-wang23@mails.tsinghua.edu.cn

Hangyu Zhang
Tsinghua University
Beijing, China
hangyu-z21@mails.tsinghua.edu.cn

Chuhan Wang
Tsinghua University
Beijing, China
wch22@mails.tsinghua.edu.cn

Jianwei Zhuge
Tsinghua University; Zhongguancun Laboratory
Beijing, China
zhugejw@tsinghua.edu.cn

Haixin Duan
Tsinghua University; Zhongguancun Laboratory
Beijing, China
duanhx@tsinghua.edu.cn

ABSTRACT

Email attachments have become a favored delivery vector for malware campaigns. In response, email attachment detectors are widely deployed to safeguard email security. However, an emerging threat arises when adversaries exploit parsing discrepancies between email detectors and clients to evade detection. Currently, uncovering these vulnerabilities still depends on manual, ad hoc methods. In this paper, we perform the first systematic evaluation of email attachment detection against parsing ambiguity vulnerabilities. We propose a novel testing methodology, MIMEminer, to systematically discover evasion vulnerabilities in email systems. We evaluated our methodology against 16 content detectors of popular email services like Gmail and iCloud, and 7 popular email clients like Outlook and Thunderbird. In total, we discovered 19 new evasion methods affecting all tested email services and clients. We further analyzed these vulnerabilities and identified three primary categories of malware evasions. We have responsibly reported those identified vulnerabilities to the affected providers to help with the remediation of such vulnerabilities and received acknowledgments from Google Gmail, Apple iCloud, Coremail, Tencent, Amavis and Perl MIME-tools.

CCS CONCEPTS

• **Security and privacy → Network security**; Intrusion/anomaly detection and malware mitigation.

*Corresponding author.

KEYWORDS

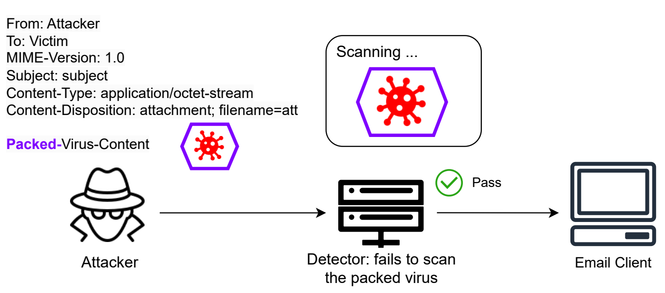
Email Security; Parsing Ambiguity; Content Detection Bypass

ACM Reference Format:
Jiahe Zhang, Jianjun Chen, Qi Wang, Hangyu Zhang, Chuhan Wang, Jianwei Zhuge, and Haixin Duan. 2024. Inbox Invasion: Exploiting MIME Ambiguities to Evade Email Attachment Detectors. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3670386

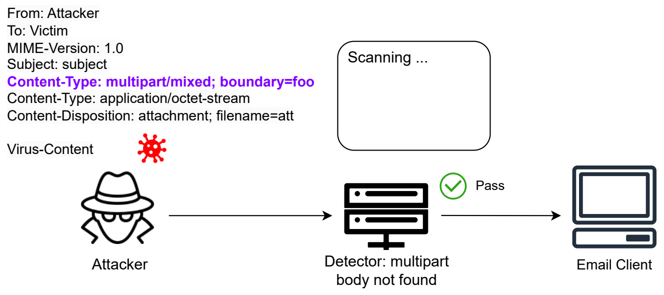
1 INTRODUCTION

Malicious email attachments have become a significant threat to cybersecurity. Those attachments contain viruses, ransomware, or phishing scripts, serving as the initial entry point for malware infections and phishing campaigns. For instance, Verizon report found that 94% of all malware is delivered by email [33], highlighting the severity of this threat vector. To combat this menace, email security appliances like malware detectors have been widely deployed [16]. Those detectors provide a crucial layer of protection to email systems by scanning incoming emails and their attachments for malicious content.

Previous studies have explored methods to evade malware detectors by altering the malware code through techniques like obfuscation, metamorphism, and binary packing [4, 28], as shown in Figure 1a. Recent findings, however, have brought to light the emergence of email parsing ambiguity vulnerabilities as a potential avenue for evading malware detection [31]. These vulnerabilities arise from discrepancies in how email messages are parsed by security detectors compared to email clients. Attackers can exploit these inconsistencies to manipulate protocol-level operations, such as the structuring and parsing of email messages, as shown in Figure 1b. This manipulation allows attackers to conceal malware within the protocol's operations, effectively bypassing malware detection systems. Despite their rising threat, these vulnerabilities are mostly


From: Attacker
To: Victim
MIME-Version: 1.0
Subject: subject
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Packed-Virus-Content
Scanning ...
Pass
Attacker
Detector: fails to scan the packed virus
Email Client

(a) Malware-level Detection Bypass


From: Attacker
To: Victim
MIME-Version: 1.0
Subject: subject
Content-Type: multipart/mixed; boundary=foo
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Virus-Content
Scanning ...
Pass
Attacker
Detector: multipart body not found
Email Client

(b) Protocol-level Detection Bypass

Figure 1: Differences in the Principle of Malware-Level and Protocol-Level Detection Bypass

identified through manual, ad hoc methods, lacking a systematic approach for detection.

In this paper, we aim to fill this gap by performing the first systematic evaluation of email attachment detection against parsing ambiguity vulnerabilities. To achieve this goal, we need to address three research questions: (1) How can we generate email samples that systematically probe potential ambiguities, given the complexity of email structures? (2) How can we effectively test cloud-based email services without dispatching an excessive volume of emails? (3) How do we detect potential attachment evasion vulnerabilities?

To tackle those questions, we propose a novel testing methodology to discover evasion vulnerabilities in email attachment detection. For the first question, we design a syntax tree-based email sample construction and mutation strategy, enabling the automatic generation of samples with structure ambiguities. For the second question, we collect a group of popular email parsing libraries as testing oracles and use them to filter those email samples with invalid structures before sending them to target email services. For the third question, we develop a testing harness to detect potential attachment evasion vulnerabilities. If an email sample can pass email attachment detection, while the mail user agent (MUA) permits users to download the original malware as an attachment, we use this disparity to identify potential evasion vulnerabilities.

We developed an automated testing tool, MIMEminer, and evaluated it on 16 content detectors, i.e., 15 detectors of popular email services and one open-source email gateway suite [3] for self-host email servers, along with 7 popular email clients. In total, we have discovered 24 evasion vulnerabilities affecting all tested email services and clients, among which 19 are newly discovered and caused 80.9% of all the evasion vulnerabilities in our test. Due to the generic nature of protocol-level evasion vulnerabilities, these vulnerabilities can be used to transmit any malware to bypass target attachment detectors. We further analyzed these vulnerabilities and identified three primary causes leading to attachment evasions: (1) Confusion over content encoding types; (2) Inconsistencies in decoding algorithms; (3) Differences in parsing malformed MIME structure; We responsibly disclosed our findings to the affected providers and received acknowledgments from notable entities such as Google Gmail, Apple iCloud, Coremail, Tencent, Amavis and Perl MIME-tools.

Contributions. In summary, we make the following contributions:

- We introduced a novel testing tool MIMEminer[1] to effectively discover email malware evasion vulnerabilities that broadly threaten email systems.
- We evaluated our tool on 16 well-known email content detectors and 7 email clients. We found 19 new vulnerabilities affecting popular email services like Google Gmail, Apple iCloud, Tencent and Coremail.
- We performed the first systematic study of email malware evasion vulnerabilities from the perspective of abnormal email structures and summarized them into three categories. We responsibly disclosed our findings to help with the remediation of such vulnerabilities and received positive feedback.

2 BACKGROUND

2.1 Email MIME Protocol

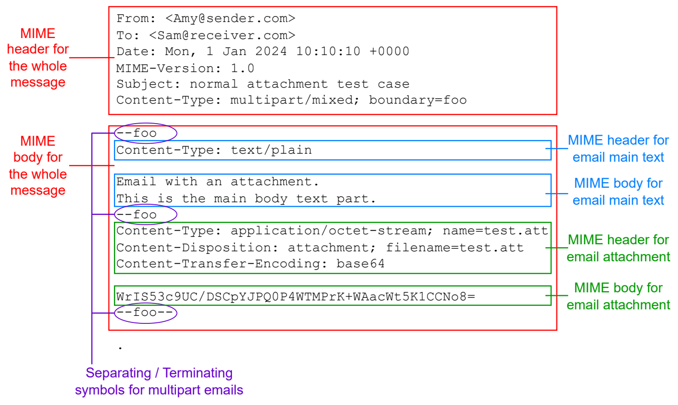
At its inception, the email system only supported the transmission of plain ASCII text content. This means that all kinds of non-English characters and binary data content (including images, sounds, videos) can not be transmitted through email. In order to meet the needs of a wider range of applications, researchers extended the original email data format specification with MIME (Multipurpose Internet Mail Extensions) [12–14, 23, 24]. It aims to realize rich content transfer functions without changing the SMTP protocol and RFC 822 [1], the initial mail content format standard.

Figure 2 shows a simple example of a MIME message. In the header section of the message, there are six header fields: From, To, Date, MIME-Version, Subject and Content-Type, where MIME-Version and Content-Type are defined by the MIME standard. After a separate blank line, there is the body of the message. The Content-Type field defines this message as a multipart type and uses the separator string "`foo`", which further divides the message body into two separate messages. The first part of the message is of type text, while the second part of the message is of type application, which is a binary file.

In the header field of the second message part, the Content-Type value is application/octet-stream, indicating that the content of the

[1]Our tool and test samples are publicly available at https://github.com/MIME-miner/MIMEminer.

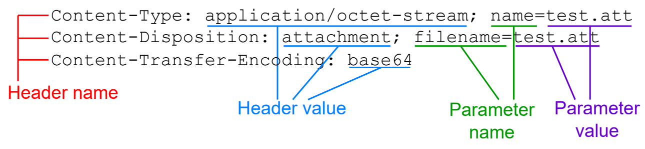
message body is a stream of binary data. The Content-Disposition header specifies that the message should be attached as part of an email message, and a parameter `filename` is used to specify the filename of the attachment. The Content-Transfer-Encoding field declares that the binary data in the message body has been base64 encoded.



```
From: <Amy@sender.com>
To: <Sam@receiver.com>
Date: Mon, 1 Jan 2024 10:10:10 +0000
MIME-Version: 1.0
Subject: normal attachment test case
Content-Type: multipart/mixed; boundary=foo

--foo
Content-Type: text/plain

Email with an attachment.
This is the main body text part.
--foo
Content-Type: application/octet-stream; name=test.att
Content-Disposition: attachment; filename=test.att
Content-Transfer-Encoding: base64

WrIS53c9UC/DSCpYJPQ0P4WTMPrK+WAacWt5K1CCNo8=
--foo--

.
```

MIME header for the whole message
MIME body for the whole message
MIME header for email main text
MIME body for email main text
MIME header for email attachment
MIME body for email attachment
Separating / Terminating symbols for multipart emails

(a) Structure of the MIME Message



```
Content-Type: application/octet-stream; name=test.att
Content-Disposition: attachment; filename=test.att
Content-Transfer-Encoding: base64
```

(b) Structure of the MIME Headers

Figure 2: An example illustrating MIME Structure

Encoding Schemes. Initially, the email transfer environment limited the transmission content to 7-bit ASCII characters. Therefore, MIME defined two encoding schemes to map arbitrary data content to the 7-bit ASCII character range, encoded data needs to be declared with the `Content-Transfer-Encoding` header to specify the encoding scheme. **quoted-printable** encoding is suitable for cases where the proportion of non-ASCII characters is relatively low and aims to preserve the readability of the original text data as much as possible. Its basic idea is to leave readable ASCII characters unchanged while encoding other octets in the form of "=" followed by hexadecimal digits (e.g., "=9A" for the octet 0x9A). On the other hand, **base64** is more suitable for general arbitrary binary data. The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. These 24 bits are treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. Each 6-bit group is used as an index into an array of 64 printable characters. When fewer than 24 input bits are available in an input group, zero bits are added and padding at the end of the data is performed using the "=" character.

2.2 Email Delivery and Attachment Detection

Figure 3 shows an example of the email delivery and detection process, as follows:

1) The sender prepares the email message through the Mail User Agent (MUA) and delivers it to the sending server. The message may contain different parts such as text body, image attachments, binary attachments, etc.

2) The sending server performs a DNS query to obtain the MX (Mail Exchange) records for the recipient domain and delivers the email to the receiving server.

3) The receiving mail server conducts content detection on incoming emails and passes the email message to the content detector, which parses the email message into various parts and invokes specific content detection engines (e.g., anti-virus engines) for malicious content scanning. If the detection engine does not identify any security-sensitive content, the email is passed down to the following components.

4) The recipient fetches the message from the receiving mail server to their MUA via POP3/IMAP and then reads the email contents, including the body and attachments.

Despite variations in the deployment of content detection gateways, we emphasize the unified threat model for detection bypass due to the independent parsing processes of *detection components* and *client components*. Given the complexity of email syntax and structure, there could possibly be a wide range of parsing inconsistencies among different components, which can be exploited for evasion attacks.

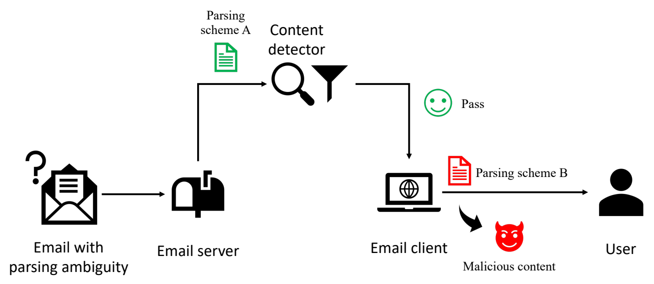


Figure 3: Email Delivery and Attachment Detection

3 OVERVIEW

In this section, we will specify the fundamentals of content detection bypass attacks based on email parsing ambiguities, including threat model, attack goals, bypass principles, possible security threats, and core research questions.

3.1 Threat Model

The attacks discussed in this paper are realized by remote operation. The attacker does not have any administrative privileges over the target mail system nor the content detection engine it employs and does not need access to information about the target mail account other than its delivery address. All the attacker should have is a controllable host capable of sending emails to the server of the target account.

The attacker's goal can be summarized as follows: an email with malicious content is delivered to a target email account without triggering the detection engine to intercept it, and the account owner receives the malicious content without receiving any security warning, therefore accomplishing bypass of the detection engine. In this work, we further specify the attack scenario as containing malicious content in email attachments, and regard the victim downloading the malicious attachments to the local device through a certain client as the destination of the attack.

Based on the previously described attacker characteristics, they can send email content with parsing ambiguity issues directly to the victim. Depending on the deployment scheme of the email system, the email may be delivered to the mail server of the cloud gateway first, or directly to the incoming mail server and then invoke the detection engine. From the email delivery process, we can notice that the parsing of emails occurs at two points: (1) the content scanning process of the detection engine (**detector side**) and (2) the process of end-users reading the content of each part of the email (**client side**). These two parsing processes are relatively independent, leaving room for parsing discrepancies that may cause security problems: i.e., the detection side fails to correctly parse the malicious content and release it, but the client eventually extracts the malicious content accurately, thus achieving detection bypass.

3.2 Motivating Example

In Figure 4 we present a real-world case we discovered. Both of the demonstrated email messages contain the same virus payload in a nested message part, but the `boundary` parameter for the inner parts are different. The sample on the left is typically intercepted by email content detectors since they are able to extract the malicious content layer by layer. However, the sample on the right can pass the detection of Microsoft Outlook because it continues searching for the non-existent inner boundary parameter and extracts nothing, while most email clients keep using the outer boundary and get the virus attachment.

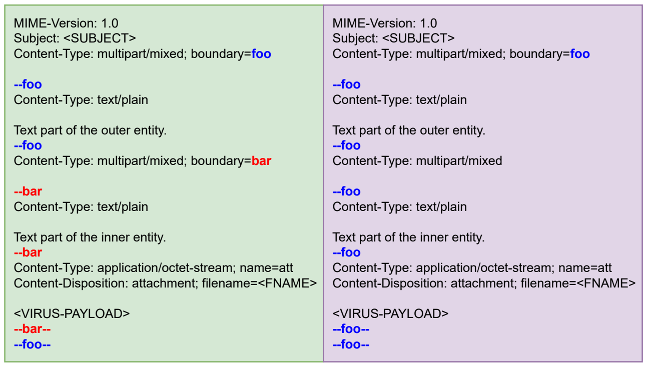
It is worth noting that the prominent feature of this attack lies in its independence from specific malicious content and its notable generality. The bypass does not involve obfuscation, encryption, or transformation of the malicious content itself; instead, it focuses on constructing malformed structures at the email level during the encapsulation process. Therefore, such a pattern can theoretically achieve the concealment of characteristics for any malicious content, regardless of its specific nature.

3.3 Research Questions

From the above description of the attack principles, the key point of these attacks is to craft malformed email structures. Such structures can lead to inconsistencies between different email parsers, thereby disrupting the judgments of malware detection engines. To efficiently find as many evasion cases as possible, we carried out our study with the following specific research questions:

RQ 1: How to achieve efficient sample generation that ensures both comprehensive exploration of ambiguity and overall structural validity?

Considering that email has complex structural specifications (e.g., widely existing nested and multi-part structures, close relationships like control and dependency between the front and back parts), it can be quite difficult to explore parsing ambiguities manually. Therefore, we designed an automated approach to generate



```
MIME-Version: 1.0
Subject: <SUBJECT>
Content-Type: multipart/mixed; boundary=foo

--foo
Content-Type: text/plain

Text part of the outer entity.
--foo
Content-Type: multipart/mixed; boundary=bar

--bar
Content-Type: text/plain

Text part of the inner entity.
--bar
Content-Type: application/octet-stream; name=att
Content-Disposition: attachment; filename=<FNAME>

<VIRUS-PAYLOAD>
--bar--
--foo--
```

```
MIME-Version: 1.0
Subject: <SUBJECT>
Content-Type: multipart/mixed; boundary=foo

--foo
Content-Type: text/plain

Text part of the outer entity.
--foo
Content-Type: multipart/mixed

--foo
Content-Type: text/plain

Text part of the inner entity.
--foo
Content-Type: application/octet-stream; name=att
Content-Disposition: attachment; filename=<FNAME>

<VIRUS-PAYLOAD>
--foo--
--foo--
```

Figure 4: A motivating example of email structure level detection bypass we discovered. The malformed sample on the right can bypass the detection engine applied in the Microsoft email service Outlook 365.

test samples. We summarize the requirements that the generation process needs to meet into two aspects: on the one hand, the results of sample generation should exhibit ample diversity to explore a wide range of abnormal sample space; on the other hand, the generated samples should adhere to basic structural norms, avoiding excessively distorted and meaningless test samples. We propose our scheme based on constructing email structures according to relevant standards and then mutating the constructed legal samples according to random or predefined mutation rules, which endows sufficient diversity of test samples.

RQ 2: How to simplify the test sample set to avoid sending a large volume of test emails towards email products and facilitate more targeted testing?

Most of the widely used email products in the real world are black-box systems, which makes us incapable of comparing the parsing differences between each other by source code analysis, nor can we obtain guidance on test sample generation based on code execution information of the parsing process. Moreover, a complete email system has various constraints like user resource limitations and spam defense, conducting black-box fuzzing directly on the email system with a large sample mass can be both costly and unethical. Therefore, we consider filtering and simplifying the test sample set prior to vulnerability mining of email products, to be specific, improving the effectiveness of test samples based on relatively simple parsing components.

The key observation behind this approach is that both the email parsers in email detectors and clients adhere to email protocol standards. Therefore, using open-source email libraries can assist in efficiently creating high-quality email test samples. This approach can reduce the testing samples for commercial email services and accelerate the testing process, as invalid samples are rejected by email libraries in our local environments before being forwarded to email services.

RQ 3: How to conduct a systematic real-product vulnerability test to detect evasion vulnerabilities?

Email products have multiple specific deployment modes in reality, creating more possibilities for security vulnerabilities related to parsing ambiguities. In this paper, we consider two types of content detection bypass patterns: one is the bypass of the native email product as a whole, in which the client used by the user is mostly the Web UI of the email product itself; the other is the bypass of different combinations of the detection side of the email system and the client side, which is manifested in the fact that the user will use an independent third-party client program with the email service. We will describe the details of vulnerability determination in Chapter 4.

4 DESIGN AND IMPLEMENTATION

4.1 Workflow

The email parsing ambiguity vulnerability mining system we designed and implemented can be divided into 3 main functional phases, as shown in Figure 5:

Sample Generation. This phase involves an email sample constructor, which automatically generates legitimate email samples according to the RFC specification, and a mutator, which randomly mutates legitimate structures to obtain malformed email samples.

Sample Filtering. In this phase, we choose a group of parsers as a *parser test group*. Based on the parsing results of different parsers on the mutated samples, we determine whether the automatically constructed email samples may indeed trigger parsing differences, and retain the set of valid test samples.

Bypass Testing. In this phase, we send valid test samples to mailbox products and email client programs respectively, and combine the parsing results of these two types of components to determine and confirm the combinations that are prone to parsing ambiguity vulnerability-based bypass attacks.

The following describes in detail how each stage works.

4.2 Sample Generation

4.2.1 *Constructing Legitimate Samples.*

In order to obtain test email samples with adequate usability, first it is necessary to automatically generate batches of legitimate email structures with an expectation to be received and parsed by most email products. For this purpose, two aspects of the rules were considered:

Syntax rules. We extract ABNF from the MIME-related RFC documents and appropriately transform them into syntax trees that can be used for sample construction, i.e., the syntax rules that guide the construction of the samples. ABNF is a paradigm that has been widely used for describing Internet protocols; similarly to the paradigm's formal characteristic of recursively defining different structure types, the syntax tree applied in the sample generation component is also characterized as a tree structure with several combinations of optional substructures, each in sequence, starting from a root node. By expanding the nodes according to the syntax tree, we can obtain a structure tree of the email, in which each leaf node corresponds to a string in the final content of the email, and the other nodes correspond to different abstract substructures in the email.

Semantic constraints. It means the control relationships claimed in the MIME standard to be observed between different parts of a complete message. While constructing the sample according to the basic structure provided by the grammar rules, additional constraints are imposed on specific occasions, which are listed in Appendix B.1.

Based on the above, the automatic constructing process of legitimate email samples can be summarized as Algorithm 1: starting from the root node, the Constructor selects the grammar tree nodes from the queue of nodes to be expanded one by one, checks their expandable child nodes, and selects the child nodes to be appended to the sample structure tree without violating the semantic constraints; when expanding to the leaf nodes (no more expandable child nodes), the Constructor fills the corresponding text content into the sample construction result according to the grammar specification constraints; and so on iteratively until the expansion is completed in all leaf nodes.

Algorithm 1 Constructing Legitimate Samples

```
Input: Grammar Tree $G$
Output: Sample Structure Tree $T$, Sample Email Content $E$
1: Initialization: Queue of nodes to expand $q$, Sample Structure Tree $T$
2: $q$.push($T$.root)
3: while !$q$.empty() do
4:     $n \leftarrow q$.pop()
5:     if $n.type$ != LEAF then
6:         $exp \leftarrow$ choose_expansion($G[n]$)
7:         for all $e$ in $exp$ do
8:             $q$.push($e$)
9:             $n$.children.append(e)
10:        end for
11:    else
12:        $str \leftarrow n$.get_content()
13:        $n$.children $\leftarrow str$
14:    end if
15: end while
16: $E \leftarrow T$.traverse_leaf_nodes()
17: return $T$, $E$
```

4.2.2 *Random Sample Mutating.*

We next randomly mutate the structure tree of legal email samples of varying morphology to explore malformed sample forms that may trigger parsing ambiguities. There are several sample mutation strategies applied in the mutator:

String Level Mutations. Operations performed on the leaf nodes of the sample structure tree that enable modifications at the emails' string level, i.e., deletion, insertion, modification of characters, etc;

Node Level Mutations. Operations performed on subtrees of the sample structure tree that enable modifications at the level of the email structure, i.e., deletion, insertion, modification, etc., of a particular mail partition;

Mutations by rule. Operations that randomly select nodes in the entire sample structure tree and execute changes according to predefined mutation rules, see Appendix B.2.

It should be noted that the above string and node-level mutations are not completely side by side, instead, they can have overlapping

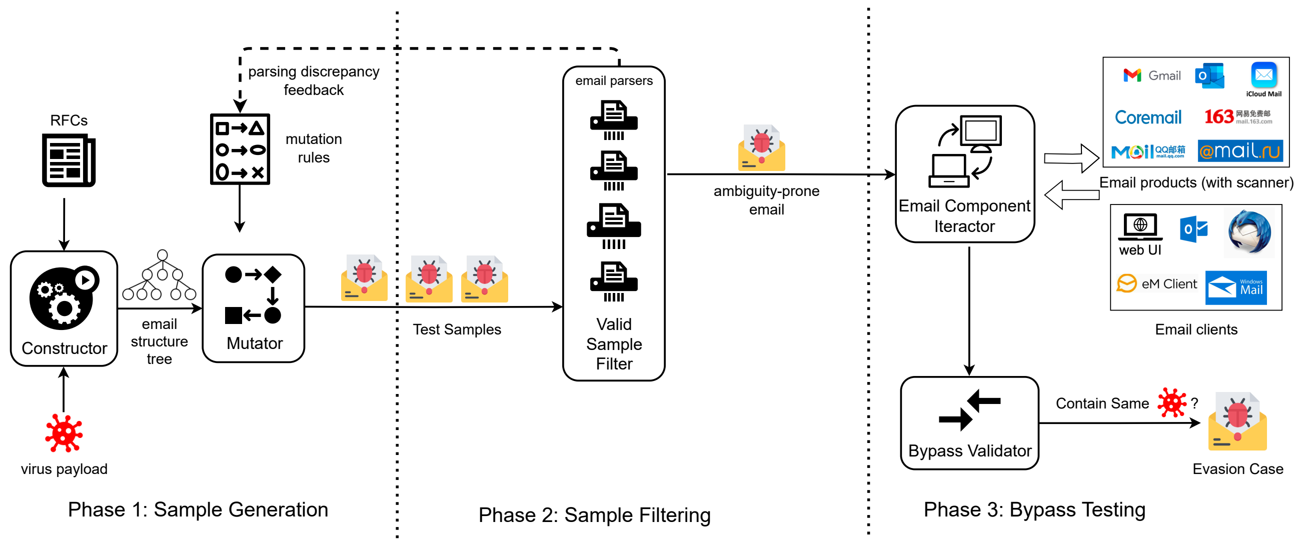


Figure 5: System Structure

mutation effects. For example, when performing node-level mutation, if the inserted node is a leaf node containing only one character, the same effect can be achieved by inserting the character at the beginning of the next leaf node through string-level mutation. This also means that the granularity of the nodes delineated in the "node-level variant" will largely determine the range of malformed samples that can be covered by the variant pattern.

In addition, the introduction of predefined mutation rules compensates for forms of variation that may be more difficult to cover in the first two schemes, but still fall within the realm of random variation, with vast room for exploration.

4.3 Sample Filtering

A large number of samples with possible parsing ambiguities can be obtained using the sample construction scheme described in the previous section, but these samples are not yet ready to be used directly for vulnerability testing, mainly for the following reasons:

- The mutation process is randomized and may result in overly malformed samples whose structures have been damaged so badly that they cannot be processed at all. Such messages can not trigger effective bypass of inspection, and therefore have no test value;
- The mutation position may appear in any part of the email, some mutation operations may change the original content, but may not affect the parsing process of various judgments, and the mutation results will not trigger parsing ambiguity;
- The sample construction module outputs a large number of samples, all of which are used in the actual mailbox product testing at a higher cost, and a large number of aberrant email-sending behaviors for the same test account may lead to blocking incoming emails.

Therefore, in order to ensure the efficiency and accuracy of actual email product testing, it is necessary to reduce the size of the testing sample set by sample filtering. We selected a group of popular libraries with mail parsing capabilities, called the parser test set. We conduct effective sample filtering through this approach mainly based on the following motivation: The implementation of the actual parsing components in both the client and detection ends of email products doesn't typically start from the most basic logic; rather, it often integrates or calls upon existing underlying tools. Therefore, triggering parsing discrepancies on mainstream library functions is likely to result in parsing ambiguities in practical email products as well. Through the parser testing group, we can effectively create a parsing simulation environment with lower testing costs.

The email samples generated in Section 4.1 are individually input into the parser testing group, invoking each parser in sequence for attachment extraction. For each sample, we compare the attachment extraction results across various parsers. We consider a sample to have successfully triggered parsing ambiguity and retain it as a valid sample when 1) there is at least one parser capable of extracting non-empty attachments, and 2) there are differences in parsing results among all parsers. Otherwise, we drop the sample.

Feedback for Mutation. Despite filtering out the email samples with a higher possibility of triggering parsing ambiguity, without proper guidance, there will be a large number of useless mutation results which can seriously reduce test efficiency. To boost the mutation process, we leverage the extraction result as the feedback to guide the later mutation. The key observation of this feedback mechanism is that *The same logic flaw appears everywhere.* To guide the mutation process, we define mutation operations as a triple: *<selector>, <operator>, <priority>*. These respectively represent which nodes will be applied to such mutation, how the mutator will mutate the nodes, and crucially, the priority of this particular mutation operation within the mutation queue. The weight can be manually or uniformly adjusted to 1 during the initial stage. The attributes for each node are predefined according to its functionality. Specifically, the initial selectors are all empty, which means they can match all the nodes. In each round of mutation, the mutator randomly selects one mutation operation based on their weights and one node that matches the selector to perform the mutation. For example, to delete a character from a newline node at random, its corresponding tripe is *.newline,remove_random_char,0.8*. The mutator will select a node that matches the selector and choose one of them randomly, and then one of its characters will be deleted. The mutated sample is then sent to the valid sample filter. Any recorded inconsistencies indicate potential inconsistencies caused by this type of mutation operation and will perform as feedback to guide the mutator, resulting in an increased priority for the mutation operation. When the current selector is empty, a new triple whose selector is the attribute of the current mutated node with the same operation and weight value will be inserted into the mutation operation queue. Conversely, if no inconsistencies arise, the priority of the corresponding mutation operation is penalized and lowered. Consequently, it can adaptively adjust the priority weights of mutation operations based on feedback from the filter to improve efficiency during the mutation process.

4.4 Bypass Testing

In actual email products, the detection end exhibits diverse forms. Given the strong integration between the detection end and the email server in many email products, testing the detection module independently can be challenging. To assess the behavior of the detection end, we conduct a series of email-sending operations toward the target detection end within the email system. The observation is focused on whether the emails can successfully pass through the detection process and be received by the user.

It's crucial to clarify that the specific goal of vulnerability exploration is to bypass the email content detection engine for inbound emails, excluding the examination of outbound emails. Therefore, the email-sending end used for testing should be as pristine as possible, possessing only the functionalities of email content encapsulation and transmission. This ensures the elimination of interference from the sender's content checks, interception, and standardization of email content.

To meet these requirements, our sending process avoids using any encapsulated email-sending software, client programs, mailbox products, or email dispatch services. Instead, we applied automated scripts that interact with the target through SMTP commands to achieve the email-sending process.

5 EVALUATION AND FINDINGS

5.1 Experiment Setup

To select email content detectors for testing, we first analyzed the list of popular email services proposed in Hu et al.'s work [15] which are equipped with anti-virus detectors. We noticed that certain email products, despite having different names and domain names, are actually owned by the same company. Additionally, these products exhibit consistent behavior in terms of interaction and parsing. Examples include Yahoo/AOL, 163.com/126.com/yeah.net, mail.com/GMX, among others. To avoid redundant data analysis, we chose one product from each set. In addition, we added one widely used gateway suite for self-hosted mail server gateway, Amavis & ClamAV, to our test list, and finally got a selection of 16 email content detectors, as shown in Table 1.

We first conducted basic behavior tests on the selected email products about the presence of virus detection. By directly sending the original virus attachment to the target mail servers and checking whether the email is rejected, discarded, isolated, or triggers a warning, we have confirmed that all the selected 16 email products have virus detection capabilities, and will take certain measures towards messages with virus attachments when no protocol-level modifications are involved. We conducted tests using the anti-virus test file EICAR [2], the exe virus WannaCry [3], and a PDF virus [4], respectively.

For email clients, we considered both the WebMail interface of the email service itself and local client programs. We selected 7 client programs based on a rank list [5] that covers major platforms. The versions of our target products are shown in Appendix A.1. We failed to connect to freemail.hu with standalone clients, and mail.com's POP3/IMAP services are included in their Premium product subscription. Therefore, we did not conduct a standalone client test for these two products.

Based on the sample construction methods outlined in 4.2, we could generate numerous malformed email samples. As is shown in Appendix A.2, we select 7 popular email-parsing libraries covering popular programming language [6] to filter mutated email samples. By applying the effective sample filtering methods described in 4.3, we finally utilized 237 email samples for testing.

Considering that some email systems may deploy domain/IP-based blocklisting for suspicious messages, we take the following measures to avoid missing emails that could bypass detection. First, we ensured the proper configuration of SPF records for our sending domain and utilized IPs with a good reputation for testing. Second, we strictly controlled the sending frequency and the number of messages within a single test to avoid triggering limitations. Third, all emails were directed to our own accounts, thus eliminating the risk of user complaints leading to blocklisting. Last and most importantly, before each experimental run, we conducted a pre-flight control test by sending a benign email to confirm its successful delivery, thereby verifying the absence of any blocklisting mechanisms in effect.

5.2 Bypass Results

In our evaluation, we generated about 5000 email samples with the Sample Generation module for the local test and retained 237 potential evasion samples after the Sample Filtering phase. We applied those samples in the remote test towards real-world email products and found that 180 of them effectively achieved detection bypass in at least one email product-client combination, yielding a success rate of 75.95%.

We found that virus detection bypass could be achieved on *all* tested 16 detectors, either by using the product's own WebMail or by using a standalone client. Out of all 128 email product-client combinations (16 detectors with 8 clients each), we discovered that

[2]https://www.eicar.org/download-anti-malware-testfile/
[3]https://www.mandiant.com/resources/blog/wannacry-malware-profile
[4]https://github.com/hacksysteam/CVE-2023-21608
[5]https://emaiclientmarketshare.com
[6]https://www.tiobe.com/tiobe-index/

102 of them exhibited email parsing ambiguities that could lead to successful detection bypasses. Detailed test result data can be found in Table 5 of Appendix A.3.

In terms of email products, Outlook has relatively fewer instances of detection bypass. This is primarily because Outlook employs strict rejection rules when receiving emails. Most emails with malformed structures are directly discarded and do not reach the recipient's server. As for email clients, the combination of Android Gmail with many detectors exhibited no bypass cases, mainly due to its lower tolerance for malformation in the attachment parsing process, leading to the failure to parse the majority of malformed structure emails.

A noteworthy result is that there are fewer or even no detection bypass issues when using the WebMail provided by the product itself. We observed 0-bypass with WebMail in 5 products (Gmail, Naver, Outlook, Yahoo, Zoho). However, this does not necessarily imply that these products have the best defensive performance against detection bypass issues. Further exploration revealed some differences in the virus detection mechanisms of Yahoo compared to other products: they accept the majority of malformed emails during the email reception stage and conduct virus detection when downloading attachments through WebMail, therefore attachments identified as security threats will fail to download. This implies that Yahoo has not effectively addressed parsing ambiguities between detection engines and clients but just deploys additional facilities to enhance protection for end users. However, this approach is far from sufficient in scenarios where third-party clients are widely used. As indicated by the test results in Table 5, the combination of Yahoo with all standalone third-party clients exhibits more detection bypass vulnerabilities compared to other products. For the remaining 4 products, even if the mail product itself employs fairly stringent checking standards, widespread inconsistencies in parsing details among diverse mail components may still result in the inability of its detector to function as expected.

5.3 Bypass Categories

After excluding known bypass methods mentioned in previous works [31], we identified 155 newly discovered samples triggering detection bypass from the 180 effective samples. Among all the 1581 evasion cases presented in Table 5, **80.9%** were triggered by our newly discovered samples, demonstrating the practicality of MIMEminer in uncovering unknown parsing ambiguity vulnerabilities. We also analyzed the effective bypass cases with reference to the existing RFC specification and found that 105 of these cases stem from vendors' failure to strictly adhere to specifications; while the remaining 75 arise from incomplete RFCs that do not specify explicit handling procedures for abnormal messages.

We summarized all effective bypass samples into 24 unique bypass methods, 19 of which were newly discovered by us. We classified these methods into three categories based on their bypass principles: (1) *Confusion over Ambiguous Header Fields*; (2) *Differences in Parsing Malformed MIME structure*; (3) *Inconsistencies in Decoding Algorithms*. Specific bypass methods of each category are listed in Table 2, and the categories that can achieve bypass on each product can be found in Table 1. In the following parts, we will illustrate the details of each category with some representative examples.

Table 1: Bypass Testing Results for Email Products. ●Vulnerable ◐Partial Vulnerable ○No Vulnerability Web. : WebMail of the Product Cli. : Standalone Client

	Ambiguous Header		Malformed Structure		Decoding Algorithm		Bypassed
	Web.	Cli.	Web.	Cli.	Web.	Cli.	
163.com	●	●	●	●	●	●	✓
Coremail	●	●	●	●	●	●	✓
Fastmail	●	●	●	●	●	●	✓
freemail.hu	●	-	●	-	●	-	✓
Gmail	○	●	○	●	○	●	✓
iCloud	●	●	●	●	●	●	✓
inbox.lv	●	●	●	●	●	●	✓
mail.com	●	-	●	-	○	-	✓
mail.ru	●	●	●	●	●	●	✓
Naver	○	●	○	●	○	●	✓
Outlook	○	●	○	●	○	○	✓
qq.com	○	●	●	●	○	●	✓
Yahoo	○	●	○	●	○	●	✓
Yandex	●	●	●	●	●	●	✓
Zoho	○	●	○	●	○	○	✓
Amavis & ClamAV	-	●	-	●	-	○	✓

Category 1: Confusion over Ambiguous Header Fields. To parse and extract specific content like attachments from email messages, the parser first needs to get basic properties about the message entity, which may include whether the message is a single-part or multi-part entity, the boundary used for multi-part entities, whether the data is encoded, and the encoding method used, etc. However, by constructing headers with ambiguity, attackers can trigger semantic gaps between different parsers. This leads to content detectors and email clients obtaining different values for the same properties, thereby achieving detection bypass.

Multiple Content-Type Headers (A3). One of the most effective methods to trigger ambiguous parsing results is by setting multiple inconsistent headers or placing multiple inconsistent values within a single header. This is particularly prominent in MIME, as its specification does not explicitly prohibit the occurrence of repeated headers, nor does it define how to handle such exceptions. In Figure 6a, we present an example discovered by MIMEminer that includes multiple Content-Type headers. In this example, detectors of Gmail, iCloud, Coremail, etc. determine the message to be a multipart entity based on the first encountered Content-Type header but fail to find the multipart boundary `foo` in the message body, thus getting no content. Contrarily, clients such as Outlook, eMClient, etc. recognize the last Content-Type header, therefore correctly extract the virus attachment.

Category 2: Differences in Parsing Malformed MIME structure. Besides confusing the detector into applying the wrong property to parse a message, an attacker can also construct malformed content for a certain structure so that the detector fails to recognize the structure correctly while the email client succeeds in doing so.

This category from the previous one primarily in two aspects: (1) the detector fails to parse a certain structure while the client can, rather than obtaining different results respectively, which also implies that such bypasses often rely on certain error-tolerant features of clients; (2) abnormal structures are not limited to header fields but may also appear in the message body.

Thanks to the mutation and filtering schemes of the MIMEminer, we were able to find many malformed samples with detection bypass capabilities. Apart from those relatively simple variations in samples such as non-standard line breaks (B3) that have been covered in studies on other protocols, our discoveries also include cases that are difficult to discover through manual auditing. The abnormality in the boundary parameter of nested message parts mentioned in Figure 4 is just one of the unusual cases (B8), and we will showcase several other detection bypass cases based on malformed structures.

Inserted Junk Characters in Headers (B2). In the example shown in Figure 6b, a byte `0x00` is inserted before the value of the Content-Transfer-Encoding header. In some languages like C/C++, this byte is used to denote the end of a string and may result in the Content-Transfer-Encoding header being truncated during parsing, making parsers unable to obtain the base64 encoding scheme. We found that detectors of Gmail, Outlook, mail.ru, etc. can't notice virus attachments in emails due to this issue, whereas clients like Thunderbird and Android Gmail ignore the `0x00` byte. Additionally, other junk characters discovered by MIMEminer that can lead to detection bypass include equal signs, quotation marks, commas, semicolons and spaces.

Empty Boundary (B4). The malformed structures causing detection bypass are not limited to appearing in header fields. As shown in Figure 6f, we specified an empty boundary parameter for a multipart entity and applied an empty string in the message body to separate different parts. According to RFC 2046, the length of the boundary string should be at least 1 byte. Detectors such as 163.com, Naver, Zoho, etc., adhere to this specification and are unable to locate a valid boundary string statement, thus cannot extract the attachment. However, Thunderbird, eMClient, and 163.com WebMail recognize the empty boundary and correctly divide different parts.

Category 3: Inconsistencies in Decoding Algorithms. MIME standards introduced two encoding schemes for binary data: base64 and quoted-printable, both defined in RFC 2045 [13]. However, due to unclear standard definitions or parsers not strictly adhering to the RFC, there are many edge cases within the encoding algorithms that can be used to trigger parsing discrepancies. By constructing non-standard encoding results, we can cause the detector to obtain incorrect decoding results, leading it to determine the content as non-malicious and achieve detection bypass.

Broken Soft-Line-Breaks in quoted-printable Data (C4). In the quoted-printable encoding scheme, the soft-line-break mechanism is used to split longer data without line breaks into multiple lines for transmission, realized by adding equal signs before line breaks (`\r\n`). In the example shown in Figure 6d, we added additional spaces between the soft-line-break symbol and the line break, disrupting the original form of the soft-line-break and violating the RFC 2045 specification, which states that there should be no spaces preceding line breaks and such spaces should be discarded. We found that detectors such as Gmail, iCloud, and qq.com did not discard spaces as specified, resulting in their decoding results containing redundant invalid content. In contrast, clients like Outlook, eMClient, and Mac Mail discarded illegal spaces and decoded the attachment content correctly.

Table 2: Categories of detection bypass and specific methods included in each. Entries marked with "*" represent our newly discovered bypass methods. "CT", "CTE", and "CD" respectively stand for `Content-Type`, `Content-Transfer-Encoding`, and `Content-Disposition` headers. "b64" and "qp" represent base64 and quoted-printable encoding schemes.

Bypass Category	Valid Bypass Methods
Confusion over Ambiguous Header Fields	(A1) multiple encoding schemes
	(A2) multiple boundary statements
	(A3) *multiple CT headers
	(A4) *abnormal capitalization of header values
	(A5) *irregular folding structure
	(A6) *overlapped header values
	(A7) comment within boundary parameter
	(A8) *comment within CTE header
	(A9) *encoded-word within headers
Differences in Parsing Malformed MIME Structure	(B1) *lack of the CD header
	(B2) *inserted junk characters in headers
	(B3) *non-standard link breaks
	(B4) *empty boundary
	(B5) *imcomplete terminating boundary
	(B6) *missing semicolon before boundary
	(B7) *redundant blank line before an entity
	(B8) *partition error in nested entities
Inconsistencies in Decoding Algorithms	(C1) inserted junk characters in b64 data
	(C2) b64 chunked encoding
	(C3) *inserted padding characters in b64 data
	(C4) *broken soft-line-breaks in qp data
	(C5) *splitted end-of-line bytes in qp data
	(C6) *abnormal capitalization of qp data
	(C7) *overlong encoded lines

5.4 Case Study

In this section, we select two representative cases to analyze the principles behind their detection bypass and introduce their real-world application effectiveness, both of which involve violations of existing RFC specifications.

RFC 2045 Violations with Widespread Impact. Due to the definition of a specific encoding alphabet, the characters in the output of base64 encoding should always belong to a given set. According to RFC 2045, if a parser encounters characters outside the defined alphabet in base64 data, it should ignore invalid characters and continue decoding. However, in our practical testing, we observed that many email components do not adhere strictly to this standard, and instead stop decoding and return the undecoded data to the caller.

In the example shown in Figure 6c (C1), multiple dots are inserted into the base64 data based on this principle. We found that many detectors, including Gmail, Yahoo, iCloud, and Yandex, do not strictly adhere to the mentioned standard, thereby failing to detect viruses in the email. On the other hand, all seven tested client programs including Thunderbird, Outlook and Android Gmail will ignore the invalid characters and extract the attachment, which demonstrates the widespread impact of such threats. We have reported this case to Gmail, and they acknowledged the severity of the issue and provided us with bug bounty rewards.

RFC 2047 Violations Due to Confusing Specification. In order to allow non-ASCII content in MIME headers, RFC 2047 proposes a mechanism called `encoded-word`, in which the transmission of arbitrary content can be achieved by appending encoded content after the character set name and encoding scheme (`B` for base64 and `Q` for quoted-printable). However, a commonly overlooked aspect is that although RFC 2047 initially asserted that this mechanism allows for non-ASCII text in *header fields*, it also includes numerous restrictions on its application, such as the prohibition of its use in *parameters of headers*, leading to considerable confusion.

In the case illustrated in Figure 6e (A9), we employ the encoded-word format to declare a boundary string for a multipart entity. We discovered that detectors of Yahoo, qq.com, Yandex, etc. don't support the boundary string in the form of encoded-word, and instead recognize the whole encoded-word `=?US-ASCII?Q?foo?=` as the boundary. In this way, they get text content ending with `--foo--`. However, the developers of Mac Mail client seemingly overlooked the restrictive conditions in the encoded-word specification. It parses the boundary parameter in the format of encoded-word and recognizes `foo` as the boundary. Consequently, the portion before the first `--foo` in the message body was discarded as a `preamble` (according to RFC 2046), and the client successfully extracts the virus attachment.

6 DISCUSSION

6.1 Responsible Disclosure

We have reported our findings to the affected providers and received acknowledgments and bug bounty rewards. The response details of the vendors are shown in Table 6.

- **Gmail**: acknowledged our report and rated the vulnerability as medium severity with a bug bounty reward of $500.
- **iCloud**: acknowledged the reported bypass vulnerability we discovered and managed to fix the issue.
- **Amavis**: arranged several discussions with us as well as the developers of the dependency Perl library `MIME-tools` [27]. The maintainers showed their concerns about malformed email structures and stated that a reasonable idea for handling ambiguous messages for a mail security component should be strict checking and exception reporting, rather than default error tolerance. Amavis and MIME-tools have currently fixed the issues in the latest versions and applied for a CVE (CVE-2024-28054) on this issue.
- **Coremail**: verified the vulnerability shortly after our report and confirmed it as Medium Risk. Coremail provided us with about $500 bug bounty rewards and has already arranged to fix the vulnerability in the latest releases. They have also carried out an in-depth discussion with us on such issues in the hope of realizing a better solution for malformed emails.
- **qq.com**: promptly responded to our report, rated it as a high-risk vulnerability, and fixed the mentioned issue. They thanked us in their hall of fame and offered us a bounty reward of about $550.
- **mail.com**: acknowledged our vulnerability report and managed to fix it. They provided us a bounty reward of €500 as well as a record in their hall of fame.
- **Outlook**: determined our finding is valid but does not meet their bar for immediate servicing. They stated that the mentioned issue has been marked for future review. However, in our latest test we found this issue has been fixed.
- **mail.ru**: indicated that the anti-spam/virus engine checks the content of the email rather than the specific files contained therein; they will inform the anti-spam team of potential content filtering bypass methods and may address this by adding static detection rules such as new feature signatures. But in practice, this does not address all parsing disambiguation detection bypass schemes in essence.
- **163.com**: disregarded our vulnerability report and didn't prepare to fix the issue.

We will continue to communicate with the vendors throughout the vulnerability disclosure process.

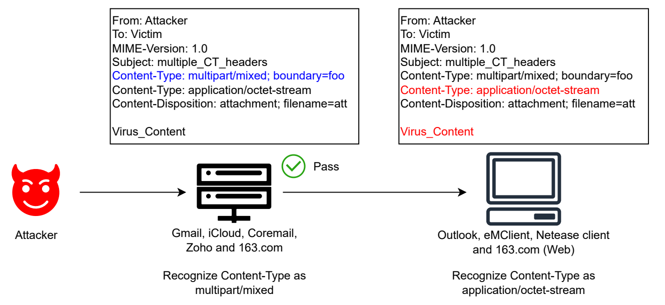
6.2 Mitigation

In terms of solving issues related to parsing ambiguities, the first thing that comes to mind is to require manufacturers to strictly comply with the latest standards and specifications in the product development process. Any deviation from the standardized format should be handled in accordance with the recommendations given in the specification. Obvious deviations from the standardized format in the absence of recommendations in the specification should be regarded as corrupted content, and email components should return or discard the message, rather than accepting it and adopting a method of disposal on its own.

The above measures are the simplest and most straightforward; However, considering the above analysis of practical reasons, this may still be too idealistic. Complete standardization of the handling of each component is indeed a suggestion, but the solution to the problem requires more institutional design. Here we think of several countermeasures to avoid the problem of parsing ambiguity without imposing consistency on the parsing components.

Stricter inspection at entry points. Instead of implementing stringent content checks on all components of the mail system, we believe that special emphasis should be placed on strict checks at the gateway entry point. On the one hand, if the inbound emails are filtered out from the first analysis and inspection, the messages with potential security problems will not reach the subsequent customer interfaces, thus not triggering the parsing inconsistency; on the other hand, although the widely-used email products have different tolerances in parsing the abnormal emails, the process of generating email contents is still in accordance with the specification, so this measure will not affect the normal sending and receiving of emails by ordinary users with email products.

In the actual product testing, the behavior patterns of Gmail and Outlook mailboxes are very much in line with the characteristics of this measure. For malformed emails, Gmail will give an abnormal response code in the SMTP session to refuse to receive the mail; although Outlook does not have the behavior of rejecting emails in

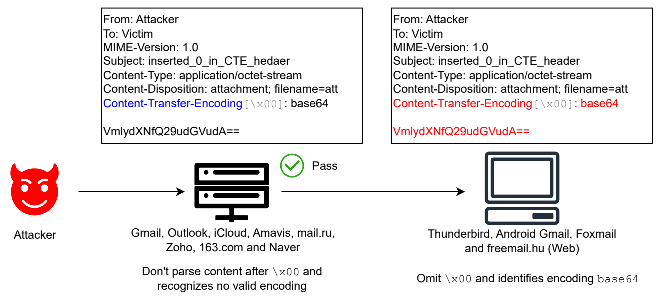


```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: multiple_CT_headers
Content-Type: multipart/mixed; boundary=foo
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att

Virus_Content
```

```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: multiple_CT_headers
Content-Type: multipart/mixed; boundary=foo
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att

Virus_Content
```

Pass
Attacker
Gmail, iCloud, Coremail, Zoho and 163.com
Recognize Content-Type as multipart/mixed
Outlook, eMClient, Netease client and 163.com (Web)
Recognize Content-Type as application/octet-stream

(a) Different preference of Content-Type headers.

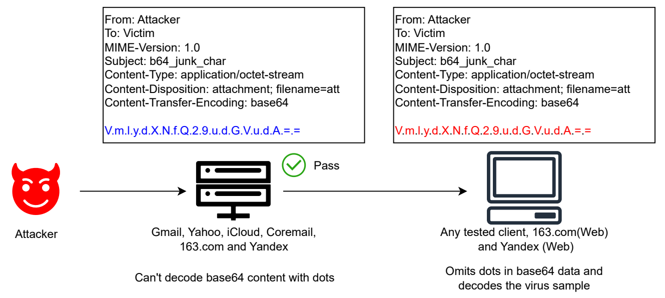


```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: inserted_0_in_CTE_hedaer
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Content-Transfer-Encoding[\x00]: base64

VmlydXNfQ29udGVudA==
```

```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: inserted_0_in_CTE_header
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Content-Transfer-Encoding[\x00]: base64

VmlydXNfQ29udGVudA==
```

Pass
Attacker
Gmail, Outlook, iCloud, Amavis, mail.ru, Zoho, 163.com and Naver
Don't parse content after \x00 and recognizes no valid encoding
Thunderbird, Android Gmail, Foxmail and freemail.hu (Web)
Omit \x00 and identifies encoding base64

(b) Differences in parsing \x00 in the header.

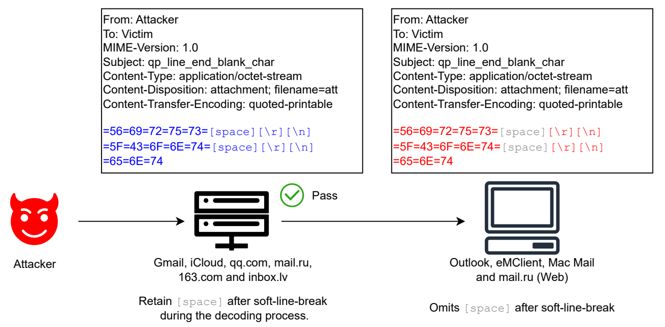


```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: b64_junk_char
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Content-Transfer-Encoding: base64

V.m.l.y.d.X.N.f.Q.2.9.u.d.G.V.u.d.A.=.=
```

```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: b64_junk_char
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Content-Transfer-Encoding: base64

V.m.l.y.d.X.N.f.Q.2.9.u.d.G.V.u.d.A.=.=
```

Pass
Attacker
Gmail, Yahoo, iCloud, Coremail, 163.com and Yandex
Can't decode base64 content with dots
Any tested client, 163.com(Web) and Yandex (Web)
Omits dots in base64 data and decodes the virus sample

(c) Differences in parsing base64 data with junk dots.

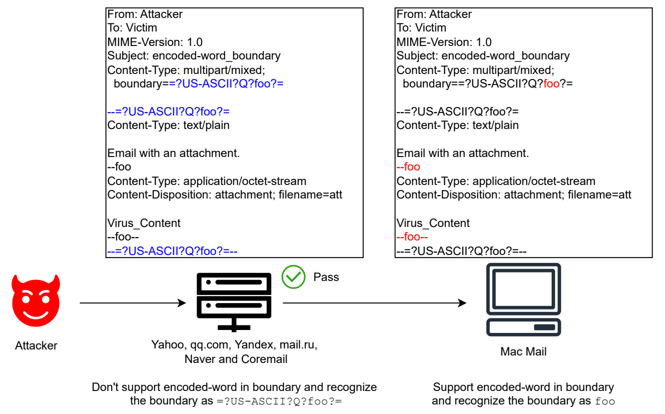


```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: qp_line_end_blank_char
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Content-Transfer-Encoding: quoted-printable

=56=69=72=75=73=[space][\r][\n]
=5F=43=6F=6E=74=[space][\r][\n]
=65=6E=74
```

```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: qp_line_end_blank_char
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att
Content-Transfer-Encoding: quoted-printable

=56=69=72=75=73=[space][\r][\n]
=5F=43=6F=6E=74=[space][\r][\n]
=65=6E=74
```

Pass
Attacker
Gmail, iCloud, qq.com, mail.ru, 163.com and inbox.lv
Retain [space] after soft-line-break during the decoding process.
Outlook, eMClient, Mac Mail and mail.ru (Web)
Omits [space] after soft-line-break

(d) Differences in parsing quoted-printable data with spaces at the end of lines

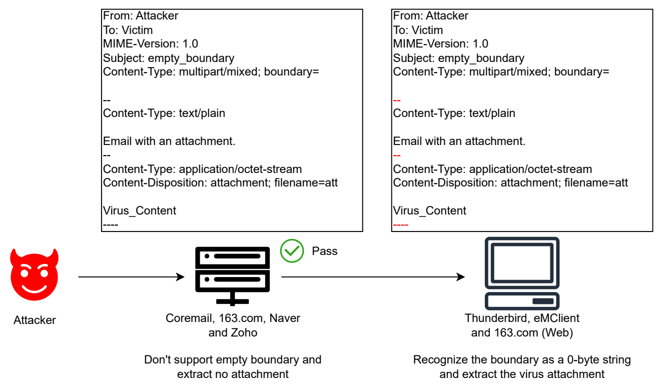


```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: encoded-word_boundary
Content-Type: multipart/mixed;
  boundary==?US-ASCII?Q?foo?=

--=?US-ASCII?Q?foo?=
Content-Type: text/plain

Email with an attachment.
--foo
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att

Virus_Content
--foo--
--=?US-ASCII?Q?foo?=--
```

```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: encoded-word_boundary
Content-Type: multipart/mixed;
  boundary==?US-ASCII?Q?foo?=

--=?US-ASCII?Q?foo?=
Content-Type: text/plain

Email with an attachment.
--foo
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att

Virus_Content
--foo--
--=?US-ASCII?Q?foo?=--
```

Pass
Attacker
Yahoo, qq.com, Yandex, mail.ru, Naver and Coremail
Don't support encoded-word in boundary and recognize the boundary as =?US-ASCII?Q?foo?=
Mac Mail
Support encoded-word in boundary and recognize the boundary as foo

(e) Differences in supporting encoded-word in the boundary parameter.



```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: empty_boundary
Content-Type: multipart/mixed; boundary=

--
Content-Type: text/plain

Email with an attachment.
--
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att

Virus_Content
----
```

```
From: Attacker
To: Victim
MIME-Version: 1.0
Subject: empty_boundary
Content-Type: multipart/mixed; boundary=

--
Content-Type: text/plain

Email with an attachment.
--
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=att

Virus_Content
----
```

Pass
Attacker
Coremail, 163.com, Naver and Zoho
Don't support empty boundary and extract no attachment
Thunderbird, eMClient and 163.com (Web)
Recognize the boundary as a 0-byte string and extract the virus attachment

(f) Differences in supporting empty boundary.

Figure 6: Samples of protocol-level detection bypass found by MIMEminer

the SMTP interaction, the emails whose contents do not conform to the norms will actually be discarded on the server.

Preference for Native Clients. According to the test results presented in 5.2, a notable observation is that the combination of email products with their own native clients (e.g., the product's integrated WebMail user interface) is less susceptible to detection bypass vulnerabilities. On the one hand, some email products (such as Yahoo) may employ additional content detection engines in the WebMail client. On the other hand, native clients developed can have better consistency in parsing with the content detection components of the product, fundamentally reducing the likelihood of exploiting parsing ambiguities to achieve detection bypass. Therefore, from a user's perspective, utilizing the product's native client whenever possible is a straightforward and effective security measure.

Parsing components ahead of inspecting engine. It is quite common for email vendors to call on third-party detection engines. The question is, pitifully email vendors do not have the ability to know whether the detection engine is consistent in its parsing principles with their own developed clients and other components related to mail functionality, and thus parsing ambiguities can occur. We believe that under the current situation, vendors can still rely on the black-box parsing behavior of the detection engine by adding parsing components ahead of the detection engine, and delivering the parsed data content to the detection engine for scanning, instead of delivering the whole mail message. Since mail vendors know the implementation details of their own clients and other components, they have the ability to implement parsing components ahead of the detection engine with identical parsing behavior, thus ensuring that the data content scanned by the detection engine is the same as that ultimately obtained by the user, and eliminating the possibility of detection bypass.

Upgrade of the current mail transport environment. The standard documents for MIME [12–14, 23, 24] were published in 1996 and have not yet been deprecated. This specification, which is more than 20 years old, is clearly not in line with the prevailing environment. Taking the encoding scheme – one of the core contents of MIME – as an example, the original intention of designing binary encoding was to make the binary data also applied to the SMTP protocol which only allows 7bit ASCII characters; however, nowadays the email data transmission environment is no longer limited to 7bit ASCII characters, contrarily 8BITMIME, SMTPUTF8 [10, 19, 22] and other mechanisms that support the transmission of arbitrary bytes have become highly popular. The idea of encoding data for transmission has lost its original application value and only brings redundant processing, which leaves room for kinds of parsing ambiguities.

Therefore, we believe that the current email content specification should be updated to remove the anachronisms that serve the rigors of transmission, including eliminating encoding and encapsulating the original data content directly in the email message. Simpler data transfer rules mean fewer possibilities for parsing ambiguities and more direct defense measures.

6.3 Limitation

In order to control the number of test emails sent to real-world email products, we introduced the sample filtering phase by locally testing on a group of mainstream email parsing libraries to retain auto-generated samples that may trigger parsing ambiguities. This design is derived from the consideration that most real-world email products are developed based on these parsing libraries, thus allowing for a lower-cost simulation of realistic environments. However, it should be acknowledged that the behavior of the actual email product may still differ from the parsing libraries, which means that the results of local testing may not fully cover bypass vulnerabilities in real-world products, resulting in certain false negatives. We set a relatively loose filtering criterion, i.e., "at least one library extracted non-empty attachments and the parsing result of all libraries differed", instead of "at least one library correctly extracted the original attachments and the parsing result of all libraries differed", in order to mitigate the effect of false negatives.

During the bypass testing phase, a significant amount of interaction with the client is required, including Web UI and standalone client programs. To closely mimic real-world email user scenarios, all our tests were conducted through graphical interfaces. Due to the diverse forms of clients and the absence of batch operation interfaces, the automation of attachment extraction becomes an inevitable challenge. While such automation is not strictly necessary for the overall research, its deficiency in the practical testing phase noticeably hinders the operational efficiency of the entire vulnerability discovery system. Addressing this limitation would contribute to further enhancing the tool's applicability and usability.

6.4 Ethical Consideration

We consider the ethical considerations in our work to primarily involve three aspects:

First, all recipient accounts used in our testing are registered by ourselves, ensuring that malicious attachments are not sent to any uninformed users. The malicious payload is confined to the attachment scope within the email and is transmitted in encoded form, posing no direct threat to the email servers responsible for receiving or forwarding. We only compare the received attachment contents from test accounts with the originally intended malicious content to verify the success of the detection bypass.

Second, our email-sending process strictly limits the sending rate. The interval between consecutive emails is maintained at 10 seconds or more, and the continuous sending of emails to the same target mail server does not exceed 30 emails at a time. All our test emails are sent from hosts within our own domain, with properly configured DNS records and reverse DNS records. The emails we send include correctly filled EHLO, Mail From, and From fields, and email service providers can get in touch with us if there are any concerns.

Lastly, both open-source and cloud email services encourage security tests through bug bounty programs. Our testing strictly follows their bug bounty rules to perform controlled experiments by sending small-scale traffic to our own accounts. And we responsibly disclosed the finding details to them.

7 RELATED WORK

7.1 Parsing Ambiguities Attacks

Jana and Shmatikov proposed two types of attacks [18] based on file parsing ambiguities against malware detectors: the Chameleon attack which obfuscates the detector's heuristics of file type determination; and the Werewolf attack which exploits format-specific file parsing differences between the detector and the OS or application. Carmony et al. investigated the PDF parser obfuscation attack techniques [5]. By comparing the differences in extraction results on the same dataset, they realized multiple obfuscation schemes on malicious PDF samples.

Apart from file parsing, parsing ambiguity in network protocols has also drawn attention from researchers. Chen et al. explored the inconsistent behaviors of different HTTP implementations towards requests with multiple Host headers, which can lead to serious attacks such as HTTP cache poisoning and security policy bypass. Their another work [6] explored the impact of email authentication based on parsing ambiguities between different components, noting that an attacker can impersonate an arbitrary sender without raising authentication anomalies. The work of Reynolds et al. [25] examines the problem of resolving ambiguities in Uniform Resource Locators (URLs) and identifies numerous deviations from parsing standards. Similar issues have been identified in web application firewalls [36] and CDN systems [7, 37].

As early as 2008, the 3proxy team published a report [2] on bypassing content inspection software, in which they listed more than 30 specific sample construction methods, including data encoding, structural damage, redundant characters, etc. Unfortunately, this report is too dated, and many of the methods mentioned are no longer applicable in the present systems. More recently, Ullrich et al published several reports [29–31] and pointed out that the complex and flexible nature of the MIME standard allows for the possibility of conflicting interpretations between different email components. They also developed an ambiguous sample generation tool MIME-is-broken [32], but this tool relies heavily on human predefined malformed rules to construct email samples and has limited scope to explore anomalous email structures. In addition, their work did not demonstrate the exact number of samples within the generated set that could effectively achieve detection bypass in real-world products.

In recent SMTP-smuggling work [20, 21], researchers discovered a class of attacks that exploited parsing inconsistencies of the end-of-data delimiters in SMTP protocol among different mail servers to smuggle or send spoofed emails while still passing the sender authentication. They have also examined the impacts of these issues by conducting tests on various real-world email products. In addition, many studies leverage vulnerabilities in SPF, DKIM, and DMARC to facilitate email spoofing attacks [34, 35].

To sum up, previous research on parsing ambiguity in email content exhibits significant shortcomings in terms of the breadth of structural exploration, the automation of vulnerability discovery, and the measurement scale of real-world products.

7.2 Automated Vulnerability Discovery and Differential Testing

Differential testing is a software testing approach based primarily on the comparison of the behavior or results produced by multiple implementation systems that are designed for similar functions. This approach is particularly useful in discovering protocol-level evasion vulnerabilities and has been successfully applied in some past research.

Chen et al. proposed mucert [9], an SSL/TLS certificate validation component discrepancy testing scheme, which diversifies seed certificates using Markov chain Monte Carlo sampling, revealing differences between certificate validation implementations. The authors also migrated similar ideas to the Java Virtual Machine vulnerability testing problem by proposing the class-fuzz fuzzy testing approach [8].

Jabiyev et al. developed a suite of differential fuzzing testers for HTTP request smuggling attacks T-reqs [17], in which a construction scheme based on context-independent syntax is designed at the sample generation level.

Foley et al. proposed the HAXSS system [11] for the cross-site scripting attack (XSS), which formalizes the problem of generating its payload as a hierarchical reinforcement learning problem and verifies the advantages of this scheme over black-box fuzzy testing systems.

Shen et al. proposed a semi-automatic detection framework, Hdiff [26], which aims at discovering possible new semantic attacks in HTTP implementations by using differential testing. Sample generation rules are extracted from HTTP specifications based on natural language processing techniques in its implementation.

Generally speaking, differential testing methods for vulnerability discovery at the email content level have not been attempted in the past. Existing testing methods in areas like HTTP are not directly applicable to email protocols because they do not adequately consider unique features of email structures such as nested entities, built-in encoding schemes, etc. Our work represents the first attempt to fill this gap.

8 CONCLUSION

Despite the increasing emphasis on content detection for sub-emails in current email systems, the inconsistency in parsing among different email components impedes the effectiveness of detection. In this work, we have designed and implemented MIMEminer for parsing ambiguity vulnerability discovery and identified 24 feasible content detection bypass methods with 19 newly discovered ones. In a broad real-world product measurement, we identified 102 combinations of detectors and clients vulnerable to detection evasion based on parsing ambiguity, with our newly discovered methods triggering 80.9% of all the evasions. Our vulnerability reports were acknowledged by Google Gmail, Apple iCloud, Tencent qq.com, Coremail, mail.com, Amavis and Perl MIME-tools.

Furthermore, the issues studied in this paper have broad potential for extension into other scenarios. Other specific applications of content detection, such as anti-spam and anti-phishing measures, also rely on the detection of specific content (including persuasive language, fraudulent URLs, etc.), therefore utilizing parsing ambiguity to achieve similar detection bypasses is feasible. We hope that our research can drive collaborative updates and iteration of standard specifications and real-world implementations, fostering a more secure email environment.

ACKNOWLEDGMENTS

We sincerely thank all anonymous reviewers and our shepherd for their insightful and constructive feedback to improve the paper. This work is supported by the National Natural Science Foundation of China (grant #62272265).

REFERENCES

[1] 1982. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES. RFC 822. https://doi.org/10.17487/RFC0822
[2] 3PROXY. 2008. Bypassing the content filtering software. https://web.archive.org/web/20221129194843/https://3proxy.ru/advisories/content.asp.
[3] Amavis. 2024. Amavis. https://gitlab.com/amavis/amavis.
[4] Hassan Asghar, Benjamin Zi Hao Zhao, Muhammad Ikram, Giang Nguyen, Dali Kaafar, Sean Lamont, and Daniel Coscia. 2022. SoK: Use of Cryptography in Malware Obfuscation. Cryptology ePrint Archive, Paper 2022/1699. https://eprint.iacr.org/2022/1699 https://eprint.iacr.org/2022/1699.
[5] Curtis Carmony, Mu Zhang, Xunchao Hu, Abhishek Bhaskar, and Heng Yin. 2016. Extract Me If You Can: Abusing PDF Parsers in Malware Detectors. In *NDSS*. https://doi.org/10.14722/ndss.2016.23483
[6] Jianjun Chen, Vern Paxson, and Jian Jiang. 2020. Composition Kills: A Case Study of Email Sender Authentication. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, USA, 2183–2199. https://www.usenix.org/conference/usenixsecurity20/presentation/chen-jianjun
[7] Jianjun Chen, Xiaofeng Zheng, Hai-Xin Duan, Jinjin Liang, Jian Jiang, Kang Li, Tao Wan, and Vern Paxson. 2016. Forwarding-Loop Attacks in Content Delivery Networks. In *NDSS*.
[8] Yuting Chen, Ting Su, and Zhendong Su. 2019. Deep Differential Testing of JVM Implementations. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1257–1268. https://doi.org/10.1109/ICSE.2019.00127
[9] Yuting Chen and Zhendong Su. 2015. Guided differential testing of certificate validation in SSL/TLS implementations. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 793–804. https://doi.org/10.1145/2786805.2786835
[10] Dave Crocker, Dr. John C. Klensin, Dr. Marshall T. Rose, and Ned Freed. 2011. SMTP Service Extension for 8-bit MIME Transport. RFC 6152. https://doi.org/

10.17487/RFC6152

[11] Myles Foley and Sergio Maffeis. 2022. Haxss: Hierarchical Reinforcement Learning for XSS Payload Generation. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE Computer Society, Los Alamitos, CA, USA, 147–158. https://doi.org/10.1109/TrustCom56396.2022.00031

[12] Ned Freed and Dr. Nathaniel S. Borenstein. 1996. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. RFC 2049. https://doi.org/10.17487/RFC2049

[13] Ned Freed and Dr. Nathaniel S. Borenstein. 1996. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045. https://doi.org/10.17487/RFC2045

[14] Ned Freed and Dr. Nathaniel S. Borenstein. 1996. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046. https://doi.org/10.17487/RFC2046

[15] Hang Hu and Gang Wang. 2018. End-to-End Measurements of Email Spoofing Attacks. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1095–1112. https://www.usenix.org/conference/usenixsecurity18/presentation/hu

[16] Fortune Business Insights. 2024. Email Security Market Size, Share & COVID-19 Impact Analysis, By Deployment (Cloud, On-Premises, and Hybrid), By Application (BFSI, Government, Healthcare, IT & Telecom, Media & Entertainment, and Others (Retail, Defense)) and by Regional Forecast, 2023-2030. https://www.fortunebusinessinsights.com/email-security-market-106607.

[17] Bahruz Jabiyev, Steven Sprecher, Kaan Onarlioglu, and Engin Kirda. 2021. T-Reqs: HTTP Request Smuggling with Differential Fuzzing. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1805–1820. https://doi.org/10.1145/3460120.3485384

[18] Suman Jana and Vitaly Shmatikov. 2012. Abusing File Processing in Malware Detectors for Fun and Profit. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12)*. 2012 IEEE Symposium on Security and Privacy (SP), USA, 80–94. https://doi.org/10.1109/SP.2012.15

[19] Dr. John C. Klensin and YangWoo Ko. 2012. Overview and Framework for Internationalized Email. RFC 6530. https://doi.org/10.17487/RFC6530

[20] Timo Longin. 2023. SMTP Smuggling - Spoofing E-Mails Worldwide. https://sec-consult.com/blog/detail/smtp-smuggling-spoofing-e-mails-worldwide/.

[21] Timo Longin. 2023. SMTP Smuggling – Spoofing E-Mails Worldwide. https://media.ccc.de/v/37c3-11782-smtp_smuggling_spoofing_e-mails_worldwide.

[22] Wei MAO and Jiankang Yao. 2012. SMTP Extension for Internationalized Email. RFC 6531. https://doi.org/10.17487/RFC6531

[23] Keith Moore. 1996. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text. RFC 2047. https://doi.org/10.17487/RFC2047

[24] Dr. Jon Postel, Dr. John C. Klensin, and Ned Freed. 1996. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. RFC 2048. https://doi.org/10.17487/RFC2048

[25] Joshua Reynolds, Adam Bates, and Michael Bailey. 2022. Equivocal URLs: Understanding the Fragmented Space of URL Parser Implementations. In *Computer Security – ESORICS 2022*, Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng (Eds.). Springer Nature Switzerland, Cham, 166–185.

[26] Kaiwen Shen, Jianyu Lu, Yaru Yang, Jianjun Chen, Mingming Zhang, Haixin Duan, Jia Zhang, and Xiaofeng Zheng. 2022. HDiff: A Semi-automatic Framework for Discovering Semantic Gap Attack in HTTP Implementations. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–13. https://doi.org/10.1109/DSN53405.2022.00014

[27] Dianne Skoll. 2024. MIME-tools. https://metacpan.org/dist/MIME-tools.

[28] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G. Bringas. 2015. SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers. In *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 659–673. https://doi.org/10.1109/SP.2015.46

[29] Steffen Ullrich. 2014. Dubious mime - conflicting content-transfer-encoding headers. https://noxxi.de/research/content-transfer-encoding.html.

[30] Steffen Ullrich. 2015. Dubious mime - conflicting multipart boundaries. https://noxxi.de/research/mime-conflicting-boundary.html.

[31] Steffen Ullrich. 2022. Mime is broken. https://2022.bsidesmunich.org/talks/001_07-W9SSVK-mime_is_broken/.

[32] Steffen Ullrich. 2022. Mime is broken. https://github.com/noxxi/mime-is-broken.

[33] Verizon. 2019. 2019 Data Breach Investigations Report. https://www.verizon.com/business/resources/reports/2019-data-breach-investigations-report-emea.pdf.

[34] Chuhan Wang, Yasuhiro Kuranaga, Yihang Wang, Mingming Zhang, Linkai Zheng, Xiang Li, Jianjun Chen, Haixin Duan, Yanzhong Lin, and Qingfeng Pan. 2024. BREAKSPF: How Shared Infrastructures Magnify SPF Vulnerabilities Across the Internet. In *NDSS*.

[35] Chuhan Wang, Kaiwen Shen, Minglei Guo, Yuxuan Zhao, Mingming Zhang, Jianjun Chen, Baojun Liu, Xiaofeng Zheng, Haixin Duan, Yanzhong Lin, and Qingfeng Pan. 2022. A Large-scale and Longitudinal Measurement Study of DKIM Deployment. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 1185–1201. https://www.usenix.org/conference/usenixsecurity22/presentation/wang-chuhan

[36] Qi Wang, Jianjun Chen, Zheyu Jiang, Run Guo, Ximeng Liu, Chao Zhang, and Haixin Duan. 2024. Break the Wall from bottom: Automated Discovery of Protocol-Level Evasion Vulnerabilities in Web Application Firewalls. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 132–132. https://doi.org/10.1109/SP54263.2024.00129

[37] Linkai Zheng, Xiang Li, Chuhan Wang, Run Guo, Haixin Duan, Jianjun Chen, Chao Zhang, and Kaiwen Shen. 2024. ReqsMiner: Automated Discovery of CDN Forwarding Request Inconsistencies with Differential Fuzzing. In *NDSS*.

A EVALUATION DETAILS

A.1 Versions of Test Targets

We list the versions of the real-world test targets in Table 3. Since most mail services do not release their system versions, the ones listed here are primarily client programs for our test.

Table 3: Versions of Test Targets

Product	Version
Mozilla Thunderbird	115.9.0
Microsoft Outlook	16.0
Foxmail	7.2.25.228
eM Client	9.2.2038
Netease Mail Master	4.17.9.1008
Mail for MacOS	16.0
Gmail for Android	2024.04.07.622678535
Amavis	2.13.0
ClamAV	1.3.1

A.2 Sample Filtering Setup Detail

As is listed in table 4, to cover common programming languages, we select 7 email-parsing libraries to filter out samples that tend to trigger parsing ambiguities.

Table 4: Email-Parsing Libraries used in Sample Filtering

Language	Email-Parsing Library
Python	email
	mail-parser
	flanker
JavaScript	mailparser
Java	Apache Commons Email
C#	MimeKit
PHP	php-mime-mail-parser

A.3 Detection Bypass Results

We present the detailed results of real-world detection bypass tests with MIMEminer in Table 5. We excluded Protonmail, Tutanota, and sapo.pt as they did not exhibit virus detection behavior. We did not test freemail.hu and mail.com using standalone clients because we could not connect to the email servers using a client.

Table 5: Number of Exploitable Bypass Samples for Each Email Product with Virus Detection

	Web Interface	Thunderbird	Outlook (client)	Foxmail	eM Client	Netease (client)	MacOS Mail	Android Gmail
163.com	14	13	15	23	15	16	17	3
Coremail	12	18	17	18	13	12	9	3
Fastmail	4	3	3	3	6	3	2	23
freemail.hu	17	-	-	-	-	-	-	-
Gmail	0	11	18	2	9	2	3	0
iCloud	9	20	15	28	21	28	12	0
inbox.lv	10	11	16	11	5	8	5	0
mail.com	4	-	-	-	-	-	-	-
mail.ru	8	23	20	4	23	4	18	17
Naver	0	24	16	14	34	2	20	42
Outlook	0	5	2	2	10	2	5	0
qq.com	2	23	15	5	25	10	20	6
Yahoo	0	78	79	73	79	82	55	6
Yandex	8	9	6	5	9	5	21	8
Zoho	0	13	12	5	27	5	18	32
Amavis & ClamAV	-	2	4	2	6	1	-	-

B DESIGN DETAILS

B.1 Semantic Constraints

The ABNF in the RFC documents defines the specification of the various email structures, but is not sufficient to specify the relationship between different structures. Therefore, MIMEminer sets the following semantic constraints during the sample construction process in 4.2.1 to ensure the legality of the automatically constructed email samples.

- For email entities of multipart types, the `Content-Transfer-Encoding` and `Content-Disposition` headers do not appear in the header field.
- For email entities of multipart types, the `boundary` parameter is set in the Content-Type header.
- In the multipart message body, the separating string between the two embedded entities matches the boundary parameter in the header of this message.
- If the `Content-Transfer-Encoding` header exists in the entity header, the data content in the entity body needs to be encoded according to the specified encoding scheme.
- For each email entity, a unique `Content-ID` value needs to be assigned.
- For the `Content-Disposition: attachment` headers, a `filename` parameter is set to specify the filename of the attachment.

B.2 Predefined Mutation Rules

While the random mutation strategies in 4.2.2 can theoretically cover a vast sample space, the introduction of predefined mutation rules that are prone to triggering parsing ambiguities can further improve the validity of the test sample. We applied the following predefined mutation rules in MIMEminer:

- Add pairs of characters to wrap part of a string, including quotes, parentheses, and angle brackets.
- Add special structure like *folding* which consists of a line-break and following null characters.
- Insert characters at regular intervals.
- Change the case of characters.
- Simultaneous mutation of boundary strings declared in the header and applied in the body.
- Apply the encoded-word form to header values.

C VULNERABILITY DISCLOSURE DETAILS

Below we show the details of the received responses for our vulnerability reports till now, including the vendor's hazard assessment of the vulnerabilities, and the current status of fixations.

Table 6: Responses of Vulnerability Disclosure

Vendor	Vulnerability Assessment	Current Status
Gmail	Medium Severity	Will Fix
iCloud	Valid	Fixed
Amavis	Valid	Fixed
Coremail	Medium Severity	Will Fix
qq.com	High Severity	Fixed
mail.com	Valid	Fixed
Outlook	Valid	Fixed
mail.ru	Disregarded	Won't Fix
163.com	Disregarded	Won't Fix