Email Transport Security in Practice

Email Client Selftest Service

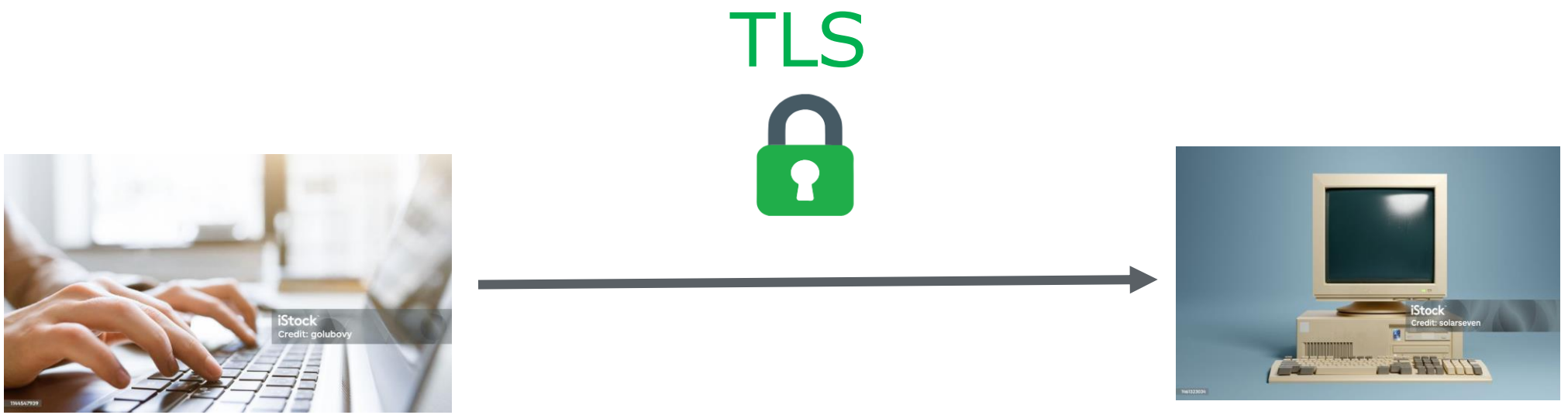
Jhannes Ernesto Reimann, Sofya Generalova

Design IT.
Create Knowledge.

www.hpi.de

Table of contents

TLS and Email protocols



- TLS secures communication on the Internet: Encryption, Integrity, Authentication
- HTTP VS HTTP**S**
- Email uses its own dedicated protocols
  - To send emails: SMTP
  - To receive emails: IMAP or POP3
- Email protocols with TLS: SMTP**S,** IMAP**S,** POP3**S**

Protocol	Normal port
SMTP	25/587
POP3	110
IMAP	143

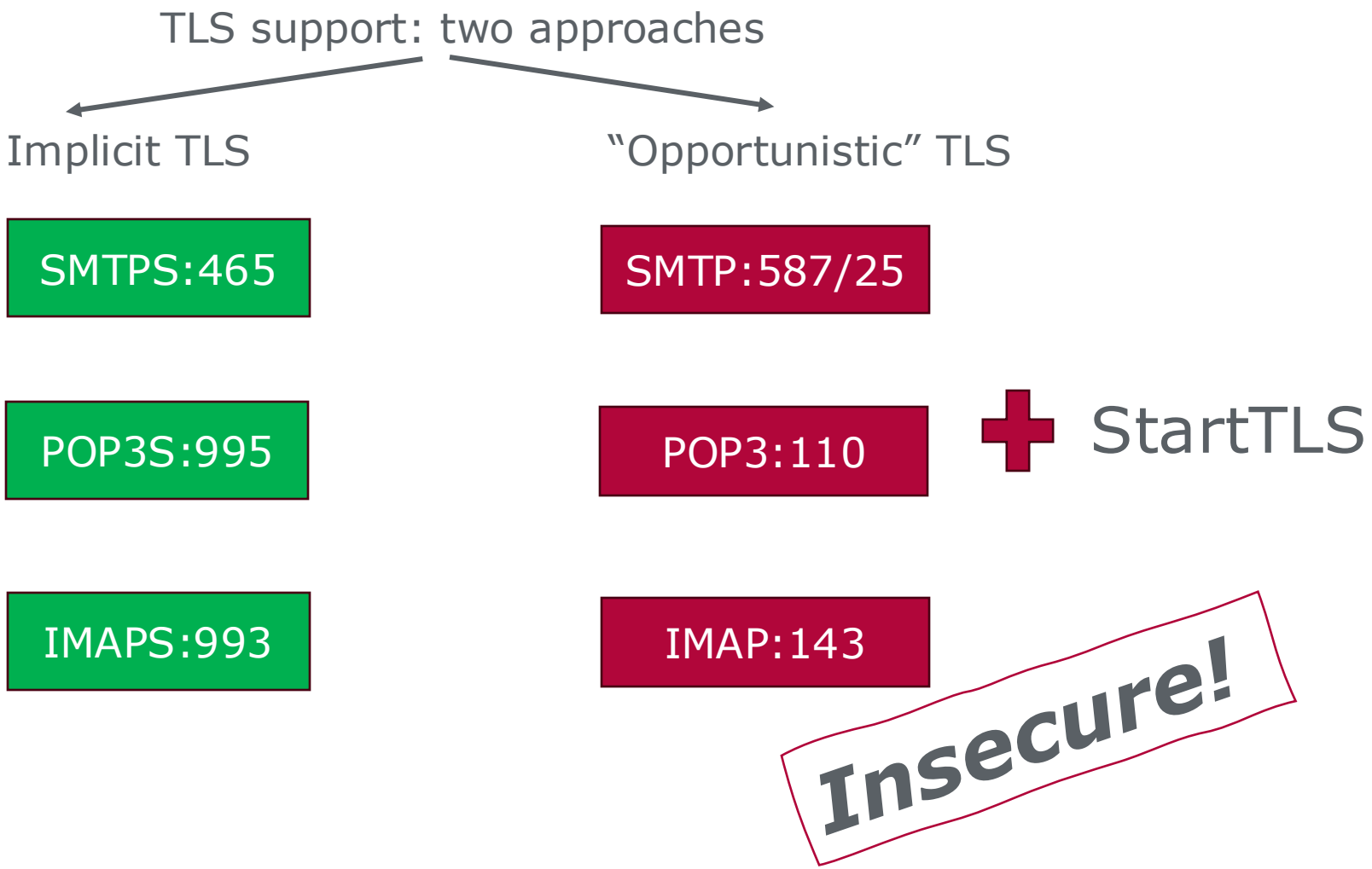
Original protocols communicate in plaintext

But: Upgrade to TLS-protected connection is possible via STARTTLS

SSL variant	SSL port
SMTPS	465
POP3S	995
IMAPS	993

Later secure protocol versions: implicit TLS support on dedicated ports

Historical context



TLS and Email in Practice: How to Configure an Email Server

SMTP Configuration with Postfix

Advice for email server admins:

1. Enable TLS-only port for client mail submission

File: */etc/postfix/master.cf*

```
# Port 465 (Implicit TLS)
smtps inet n - y - - smtpd
-o syslog_name=postfix/smtps
-o smtpd_tls_wrappermode=yes
-o smtpd_sasl_auth_enable=yes
```

SSL variant	SSL port
SMTPS	465

2. Often support for legacy ports is needed, ensure TLS is enforced

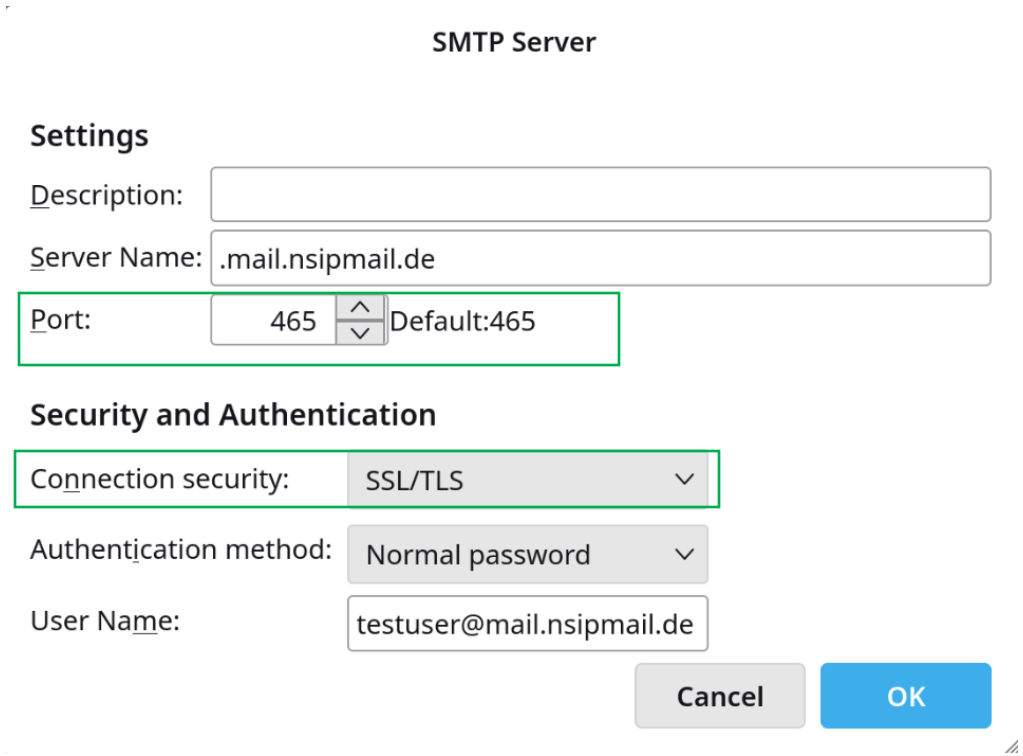
File: */etc/postfix/master.cf*

```
# Port 587 (STARTTLS)
submission inet n - y - - smtpd
  # -o smtpd_tls_security_level=may
  # -o smtpd_tls_auth_only=no
  -o smtpd_tls_security_level=encrypt
  -o smtpd_tls_auth_only=yes
```

Protocol	Normal port
SMTP	25/587

TLS and Email in Practice: Client-Side perspective

In Thunderbird:



Usually Email Clients automatically suggest TLS configuration

Advice to users:

- Use secure email clients
  - Example: Microsoft Outlook mail client only supports SMTPS
- Do not change TLS settings with no good reason

Problem: Users do not control how email clients enforce TLS internally

A client may: fall back to insecure connection on disruption, automatically apply insecure options with autodetection mechanism

TLS and Email in Practice: Server and Client Side

Summary

Email Server Admins should:

- Allow Email clients to submit mail on TLS-only port
- *For legacy ports enforce TLS-only connections*

-> Email Server Configuration Scanner for admins

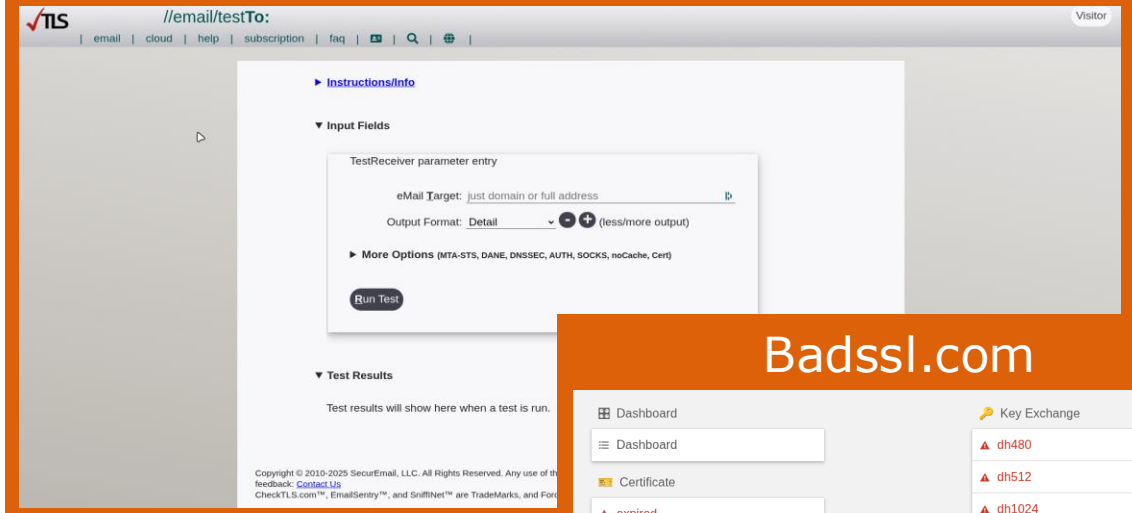
Email Users should:

- *Select trusted Email Clients*

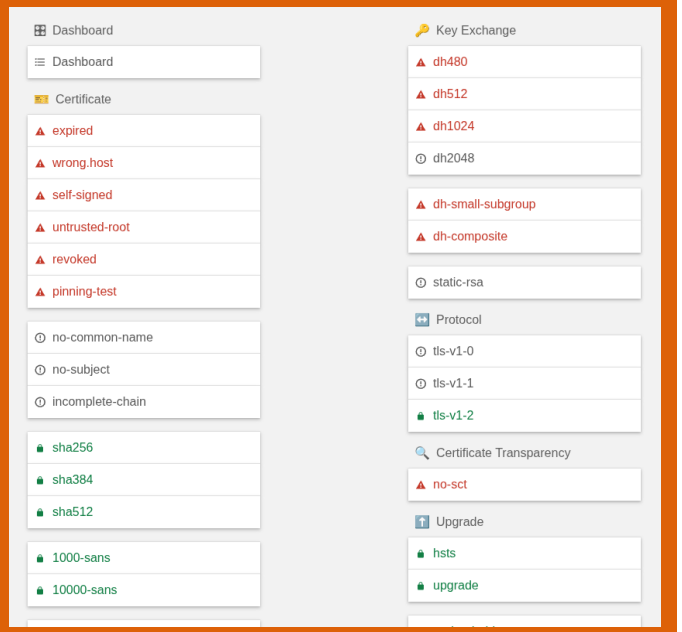
-> Email Client Selftest Service for users

Current State of Email Security Testing

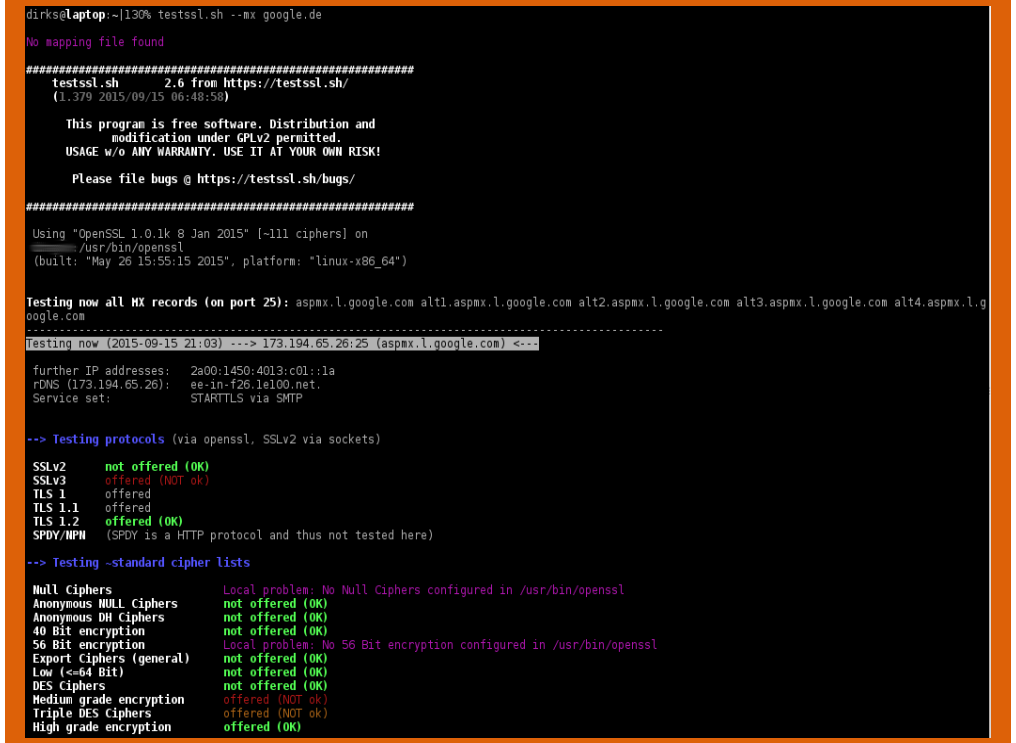
checktls.com



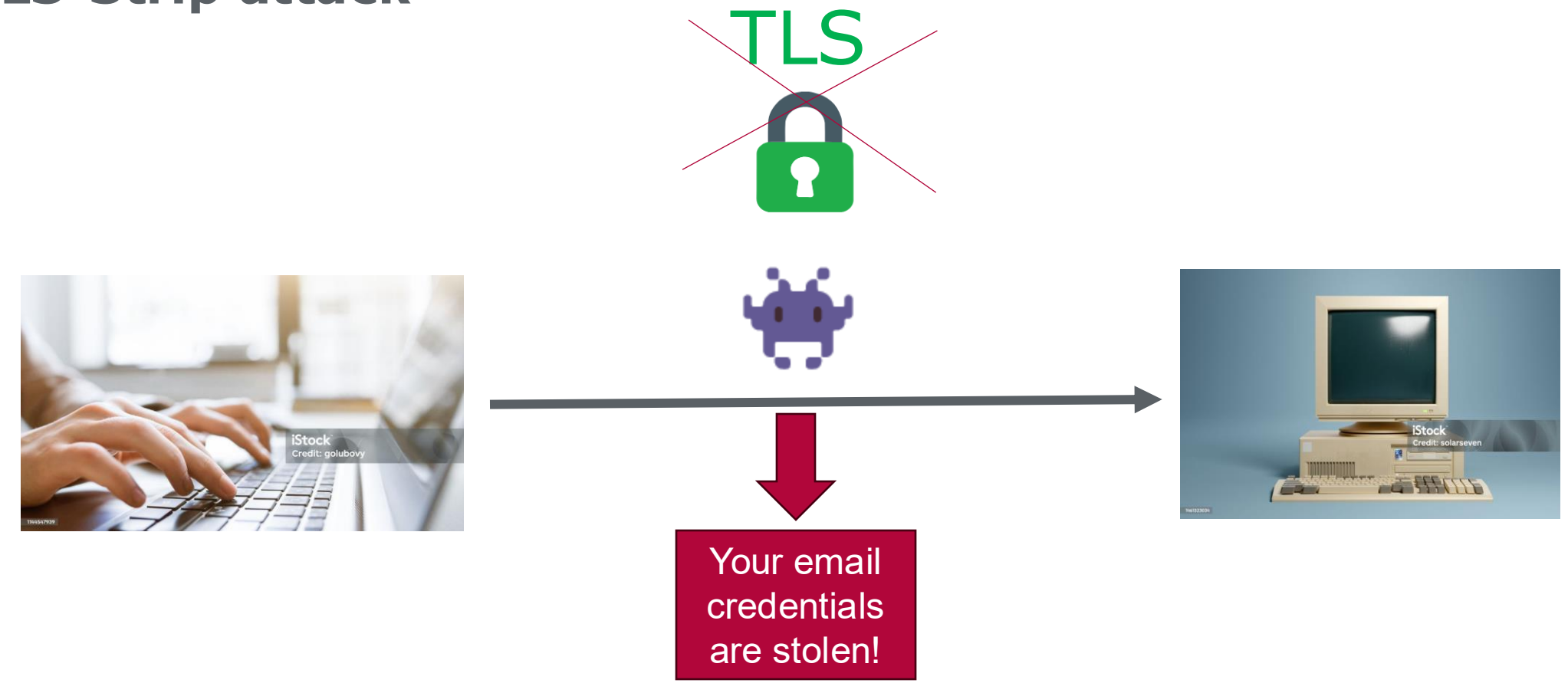
Badssl.com



Testssl.sh

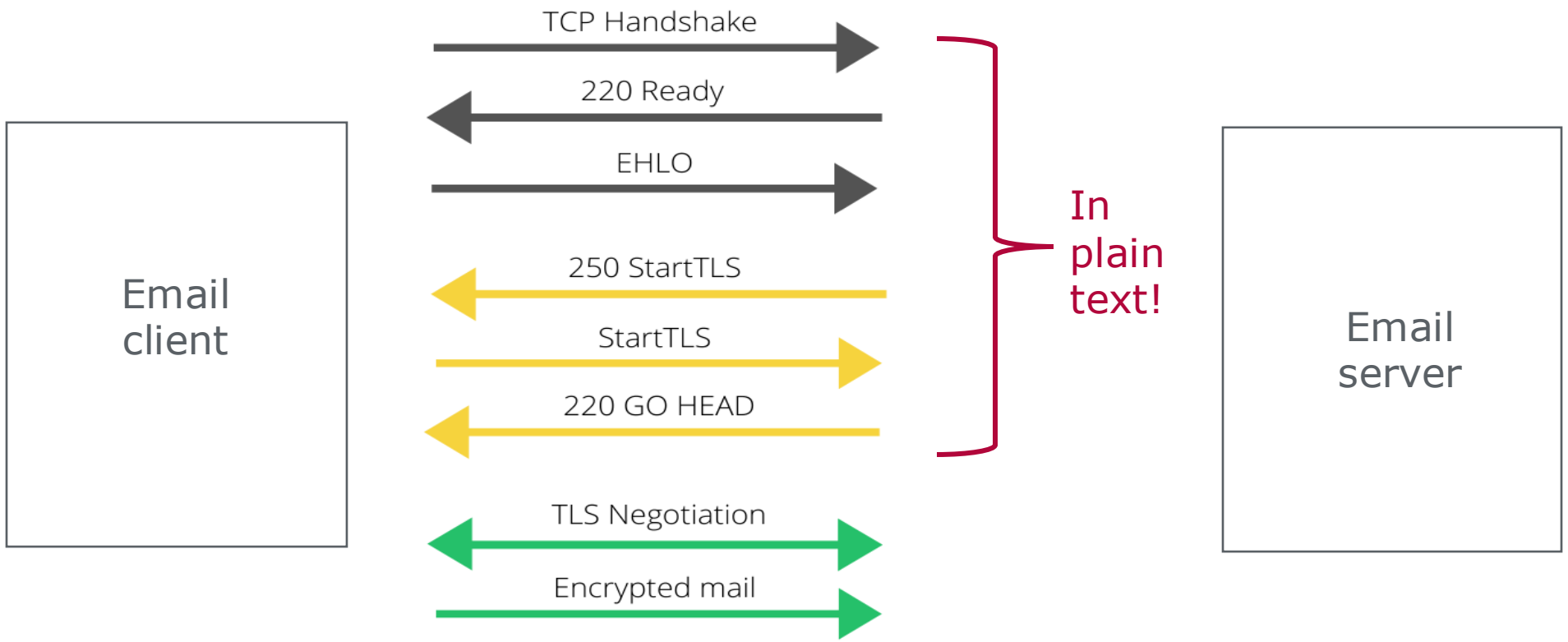


TLS-Strip attack



How Is That Possible?

Vulnerability details: Opportunistic TLS



SMTP protocol with StartTLS

https://www.twilio.com/en-us/blog/insights/what-is-starttls

An active attacker on the network can strip StartTLS and force no-TLS

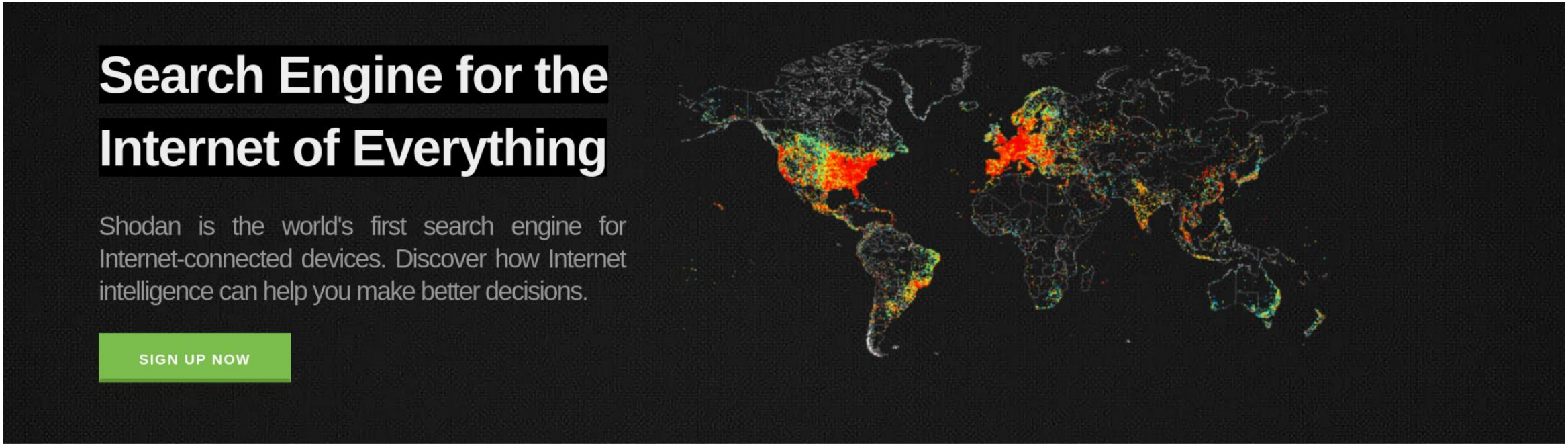
if email server and client don't explicitly disallow it

Security downgrade is possible if **email server** and client don’t explicitly disallow it

Vulnerable Servers

How many email servers would accept plaintext?

Shodan



- A web-based search platform for Internet connected devices
- We query Shodan API - https://api.shodan.io/shodan/host/count
- Example query: port:587 ("ESMTP" OR "EHLO" OR "250-")
- Shodan searches its database, we do not connect to actual servers on the Internet

Passive Measurement with Shodan

Idea: scan Internet for email servers that advertise plain auth

SMTP example:

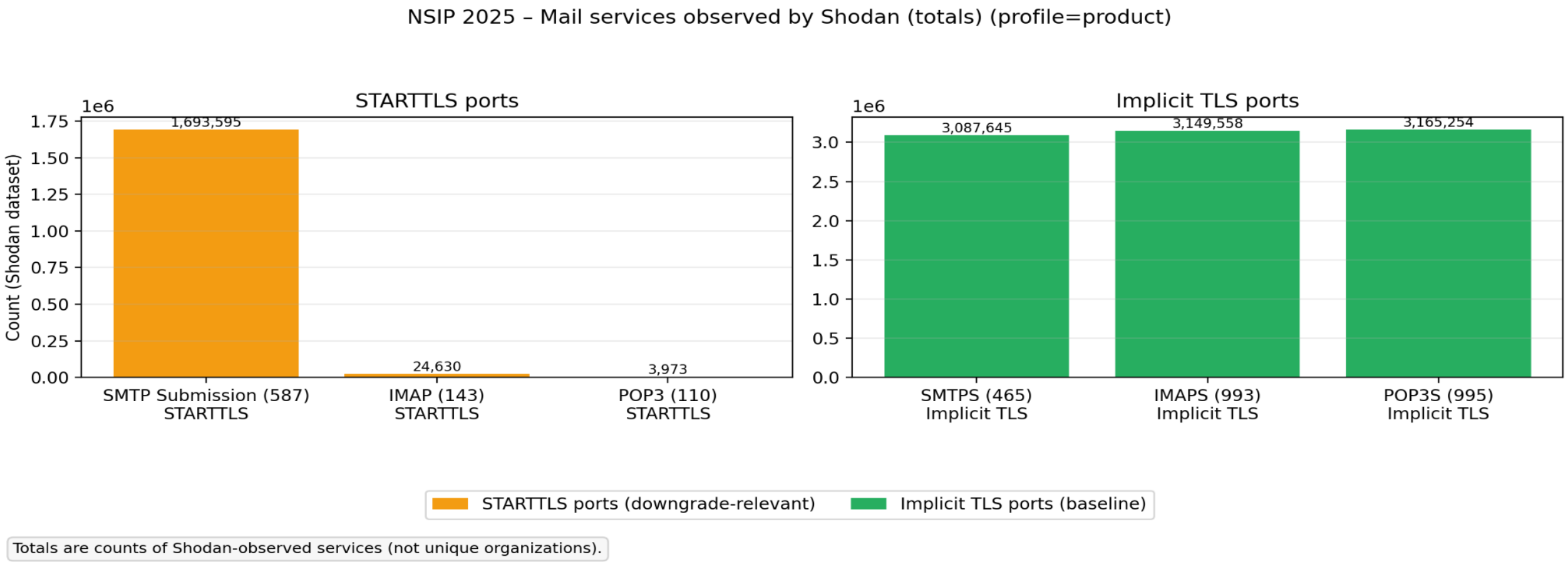
1.Client requests capabilities from server	EHLO test.com
2.Server lists capabilities	250-mail.nsipmail.de 250-PIPELINING 250-SIZE 10240000 250-VRFY 250-ETRN 250-STARTTLS **250-AUTH PLAIN LOGIN** ...

potentially effective before TLS →

This does not guarantee server's vulnerability:

Client tries to login	C: AUTH PLAIN AGFsaWNlAHNlY3JldA==
Server rejects	S: 530 5.7.0 Must issue a STARTTLS command first

Passive Measurement with Shodan: results



Email Server Configuration Scanner

- Bash script
- Postfix (SMTP) & Dovecot (IMAP, POP3)

Detects and issues warnings if:

- TLS is optional instead of mandatory
- insecure authentication options (plain, login)

Multiple config file support:

• Dovecot: Handles split(*conf.d/*) and monolithic(*dovecot.conf*) configurations

• Postfix: Supports *main.cf* and *master.cf*

```
ubuntu@mail:~$ ./server-checker-for-admin.sh
=== Mail Server Vulnerability Audit ===
This script checks Postfix/Dovecot configs for vulnerability to STARTTLS downgrade.

[Postfix] smtpd_tls_security_level=may
  Reason: TLS is optional; STARTTLS stripping is possible
  Recommendation: smtpd_tls_security_level = encrypt
  File: /etc/postfix/main.cf

[Postfix] smtpd_tls_auth_only=no
  Reason: AUTH allowed before TLS negotiation
  Recommendation: smtpd_tls_auth_only = yes
  File: /etc/postfix/main.cf

[Postfix]   -o smtpd_tls_security_level=may
  Reason: submission service allows STARTTLS downgrade
  Recommendation: smtpd_tls_security_level = encrypt
  File: /etc/postfix/master.cf

[Postfix]   -o smtpd_tls_auth_only=no
  Reason: submission service allows AUTH before TLS
  Recommendation: smtpd_tls_auth_only = yes
  File: /etc/postfix/master.cf

Postfix config documentation:
  main.cf -> man 5 postconf, https://www.postfix.org/postconf.5.html
  master.cf -> man 5 master, http://www.postfix.org/master.5.html

[Dovecot] disable_plaintext_auth=no
  Reason: Allows cleartext authentication
  Recommendation: disable_plaintext_auth = yes
  File: /etc/dovecot/conf.d/10-auth.conf

[Dovecot] ssl=yes
  Reason: TLS is optional; downgrade possible
  Recommendation: ssl = required
  File: /etc/dovecot/conf.d/10-ssl.conf

[Dovecot] auth_mechanisms = plain login
  Reason: Plain or LOGIN auth enabled — safe only with mandatory TLS
  Recommendation: Use mandatory TLS with PLAIN/LOGIN or disable them
  File: /etc/dovecot/conf.d/10-auth.conf
```

Security downgrade is possible if email server and **client** don’t explicitly disallow it

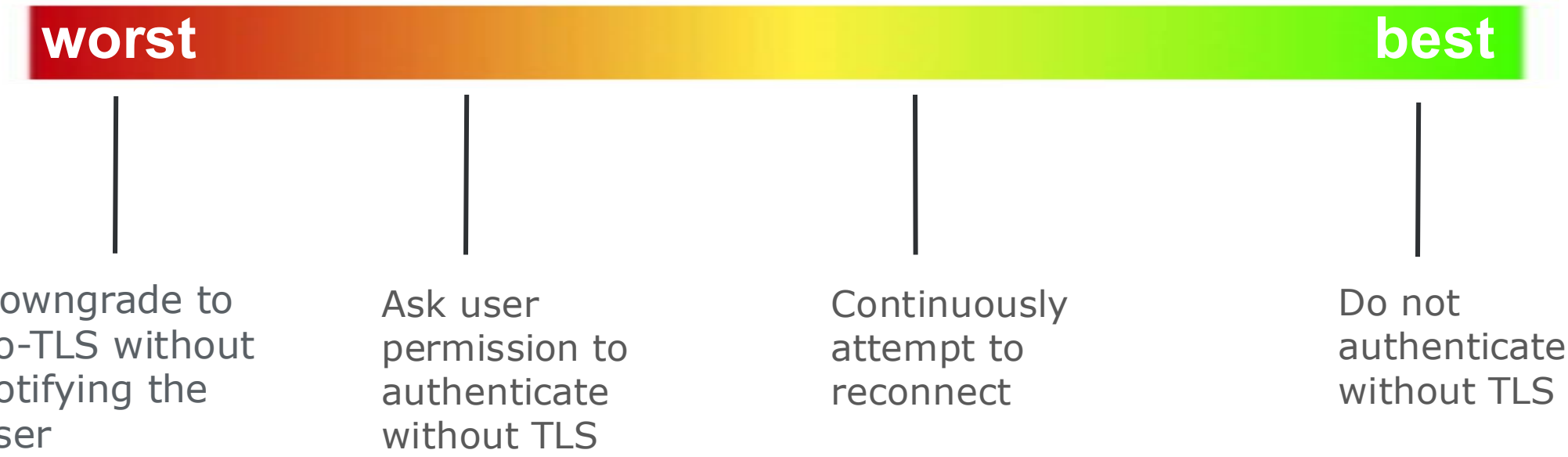
Vulnerable Clients

How many email clients would authenticate in plaintext?

A Multifaceted Study on the Use of TLS and Auto-detect in Email Ecosystems (2025)

Email client behavior*:

*can differ depending on email protocol and TLS downgrade test case

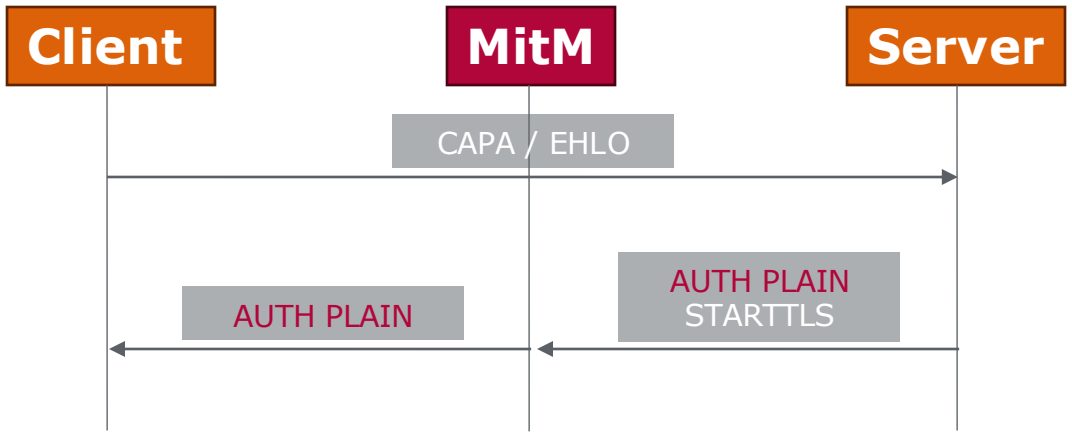


19 out of 49 tested clients may silently downgrade to no-TLS

A Multifaceted Study on the Use of TLS and Auto-detect in Email Ecosystems (2025)

Testing email client behavior: 4 variations of disruptions

T1:
"classic" strip
StartTLS test



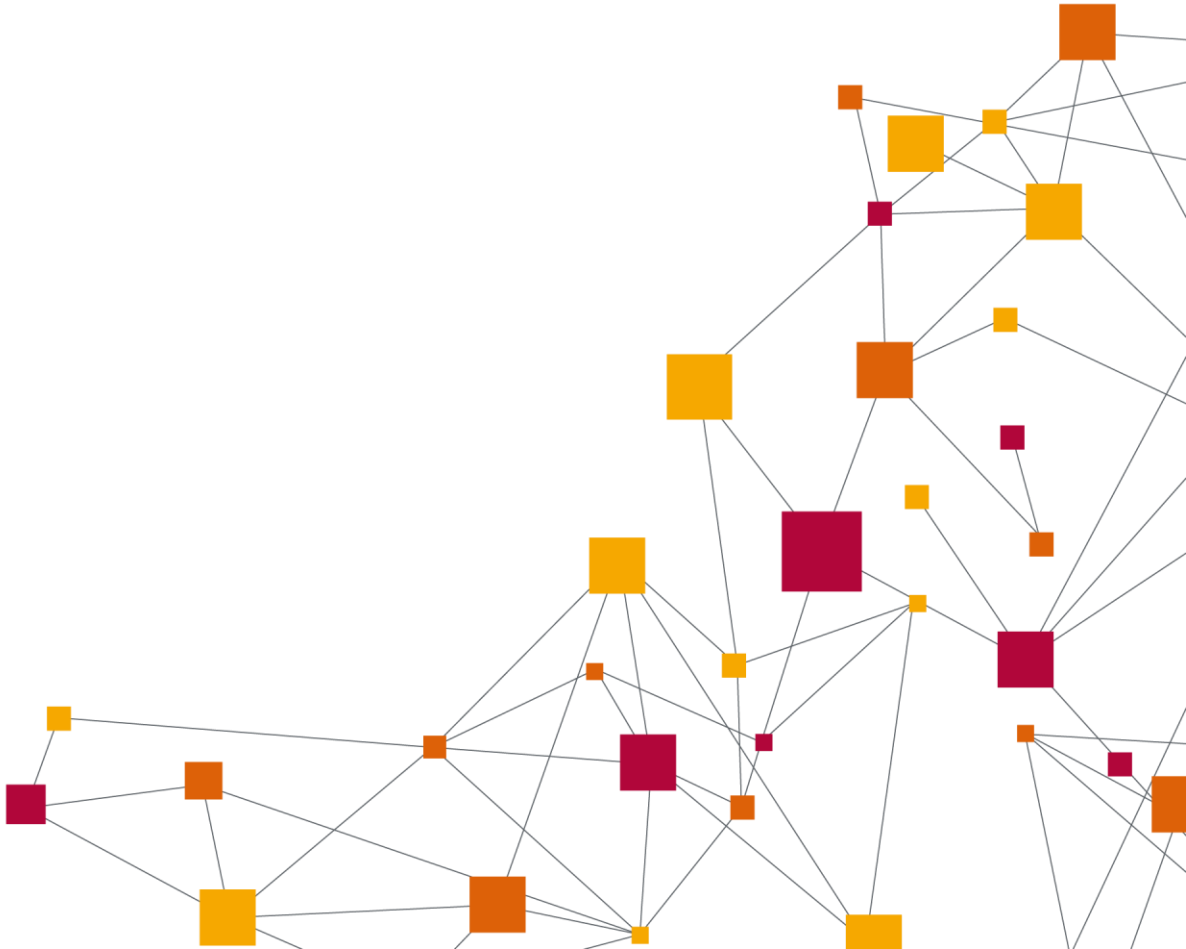
T2: disrupt during TLS handshake
T3: disrupt after client selects STARTTLS
T4: disrupt ongoing TLS session

Is your email client vulnerable?

Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Selftest Service

A way to test your email client in use.



Design IT.
Create Knowledge.

www.hpi.de

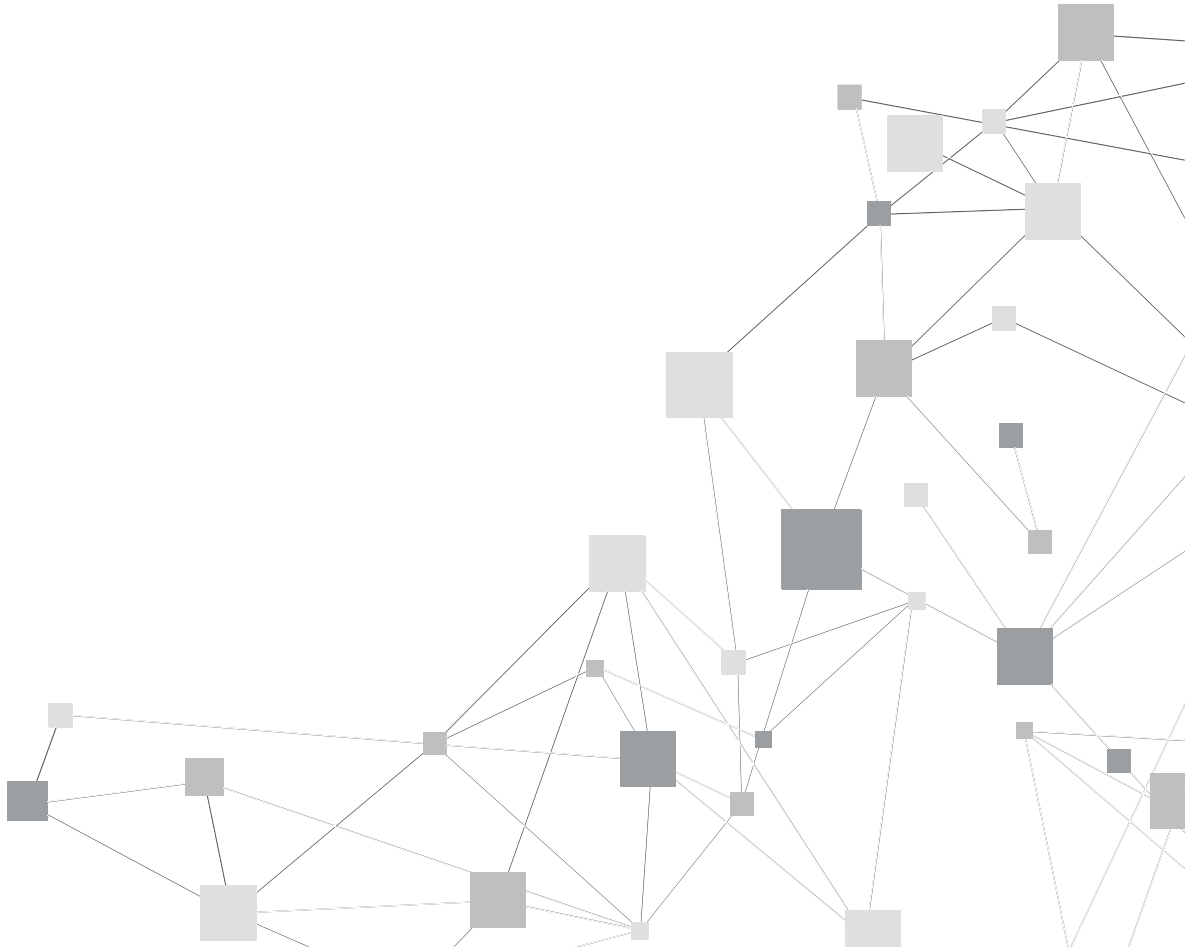
What did we build?

- Public, server-side **mail client self-test** for **SMTP + IMAP**
- Tests whether a client can be coerced into **plaintext authentication** when TLS protection is disrupted
- Implements **baseline + T1–T4** behaviors (paper-inspired)

Why did we build it?

- Paper setup assumes a **MITM/lab environment** (hard to reproduce for normal users)
- We want a **“click → configure account → observe result”** workflow
- No local proxy, no custom CA, no special network setup required

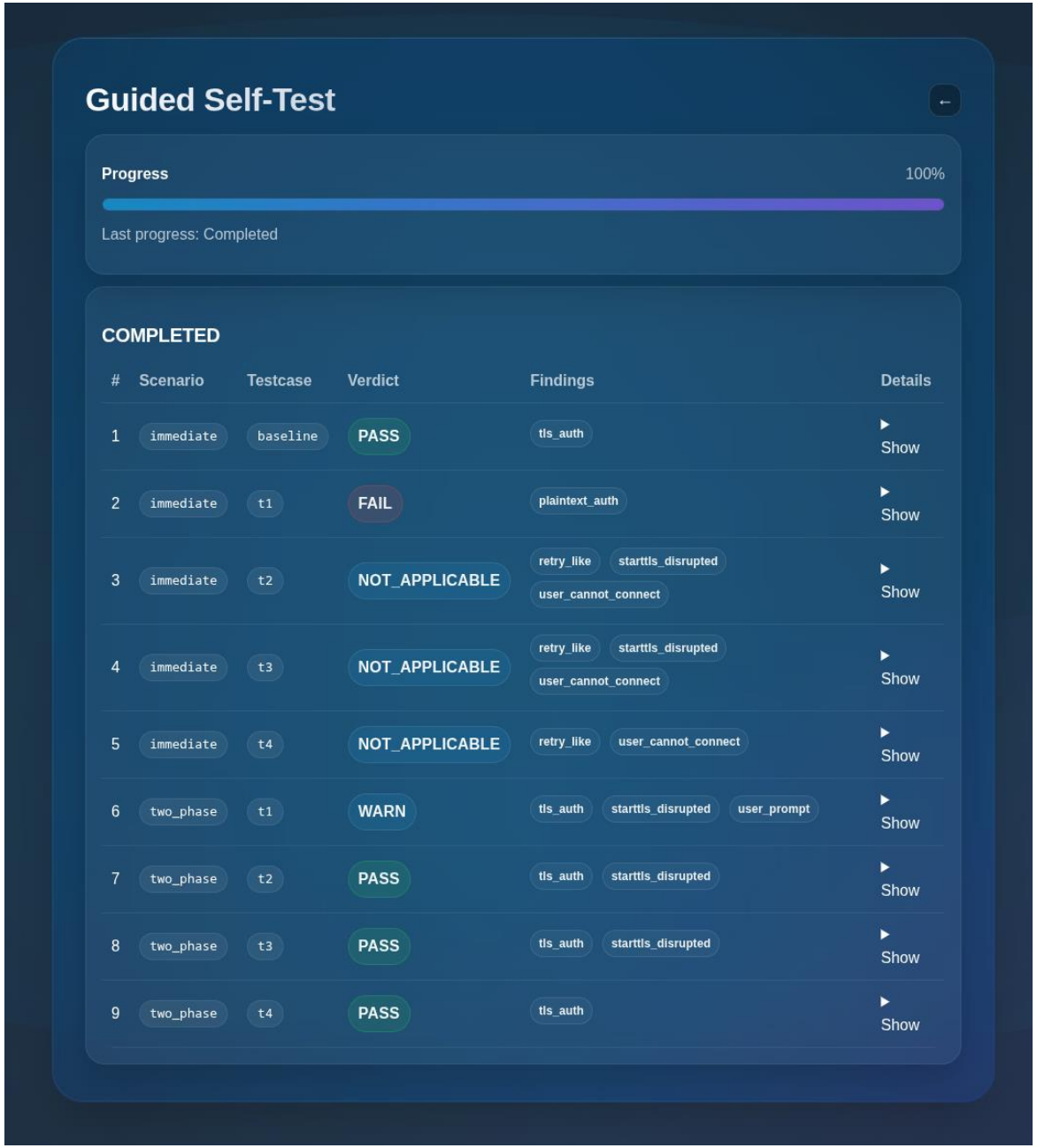
Demo



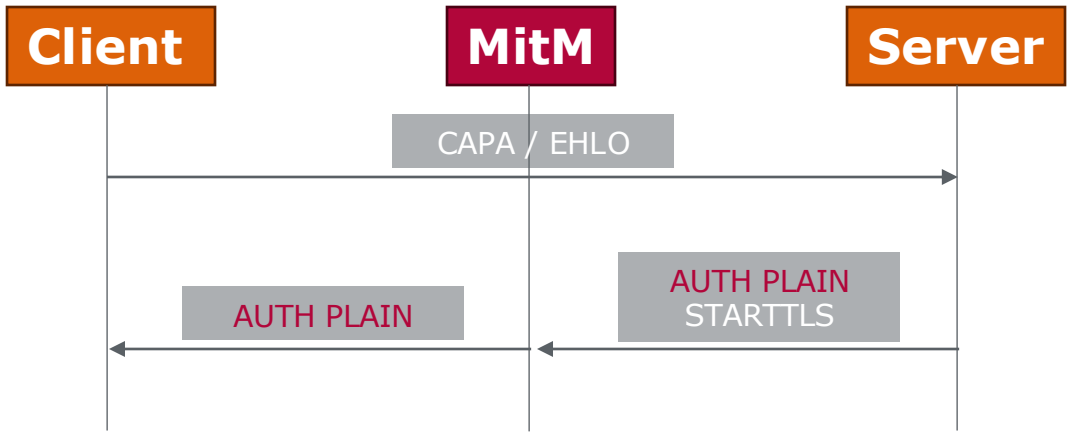
Design IT.
Create Knowledge.

www.hpi.de

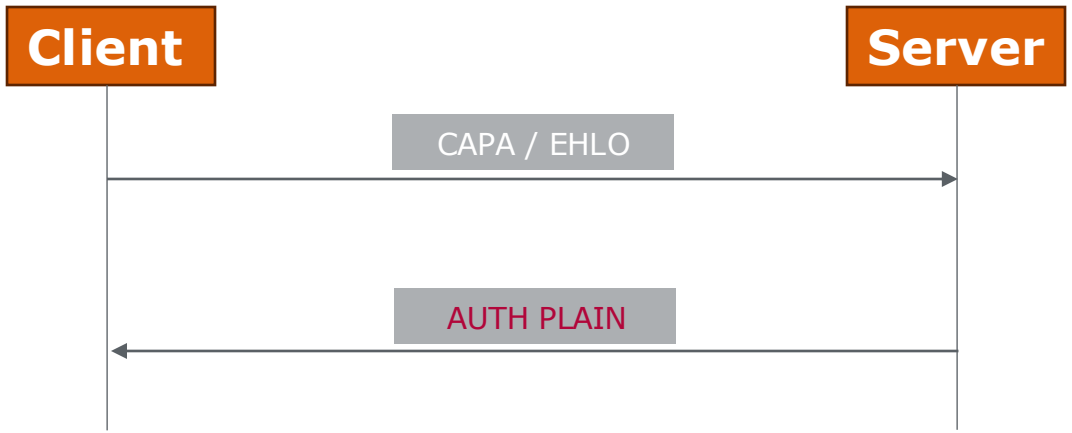
Result example



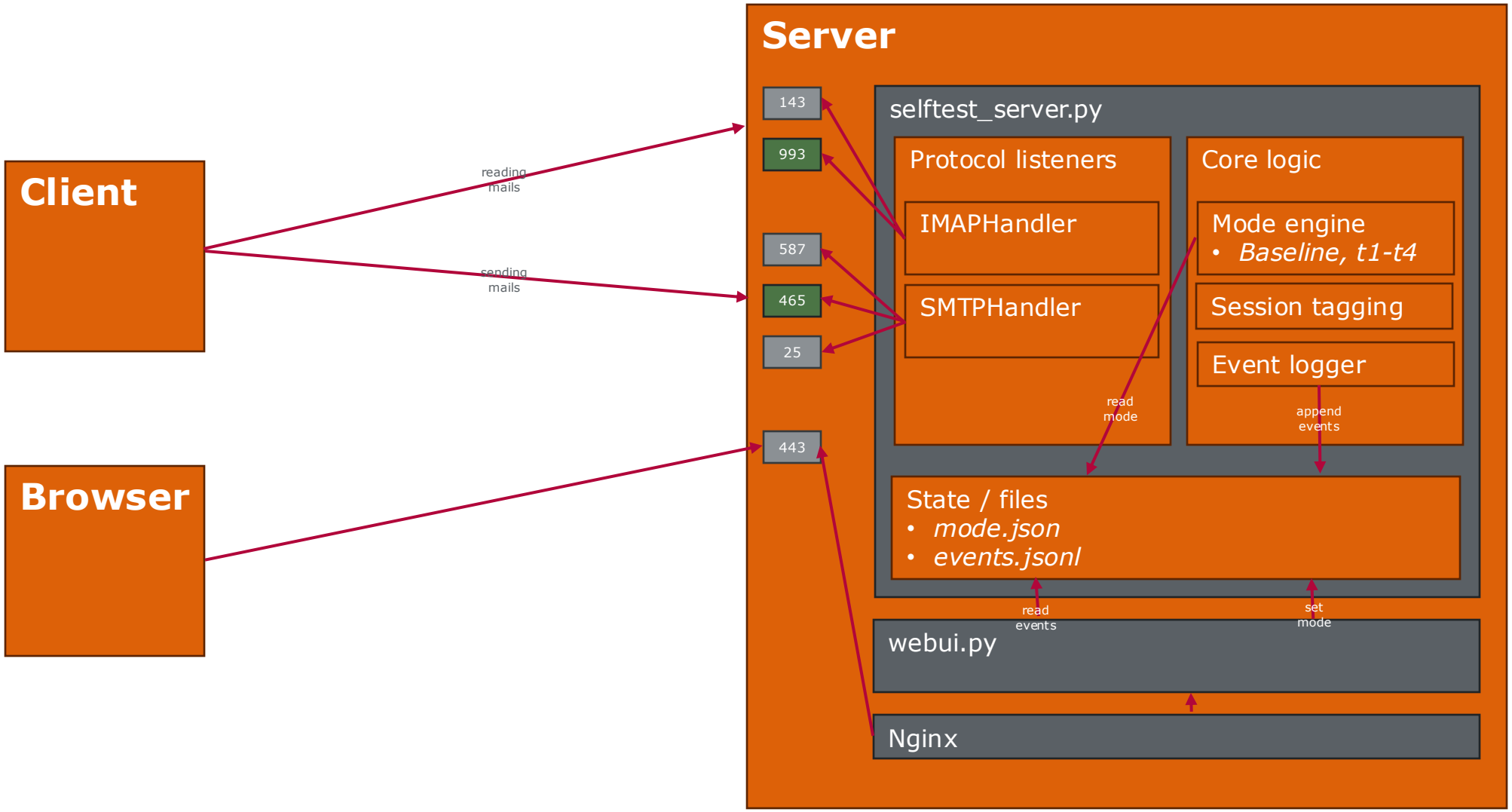
"Simulation" Diagram of T1



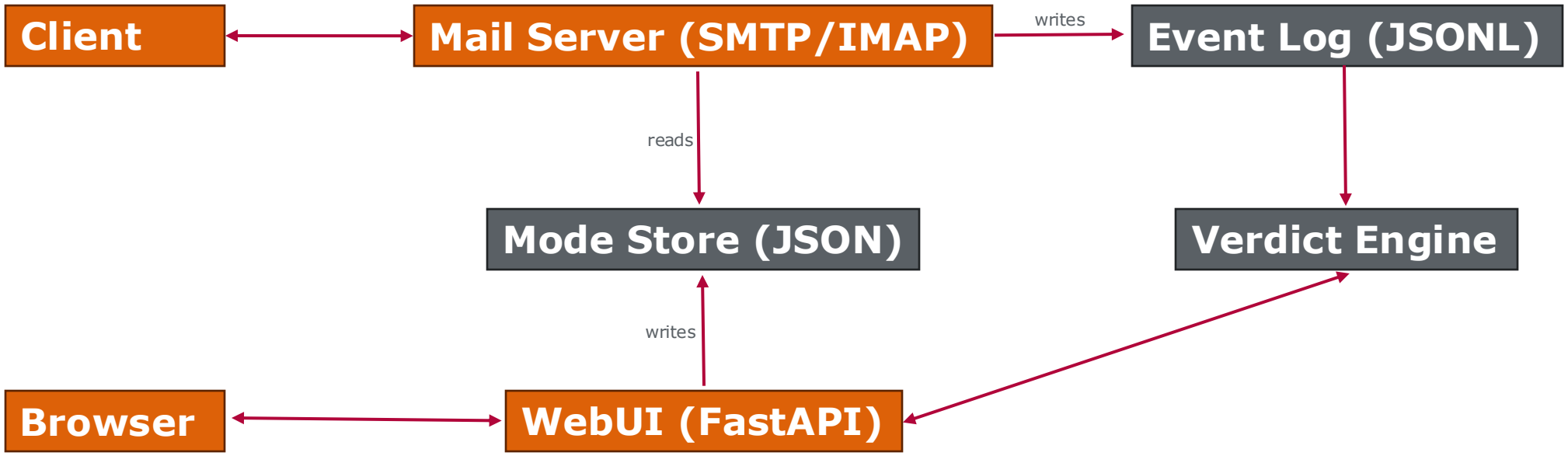
"Simulation" Diagram of T1



Selftest-Service architecture



Selftest-Service architecture



Handler Architecture – Code Overview

```python
class SelfTestSMTPHandler(socketserver.BaseRequestHandler):
    def handle(self) -> None:
        # 1. Load mode decision
        dec = _decide_mode(self.server.mode_store_path, client_ip)
        # 2. Block implicit TLS (T1-T4: ports 465/993 → disconnect)
        if tls_active and _should_block_implicit_tls(dec.mode, server_port):
            _log_event( ... , event="disconnect", reason="implicit_tls_blocked")
            try:
                sock.shutdown(socket.SHUT_RDWR)
            except Exception:
                pass
            try:
                sock.close()
            except Exception:
                pass
            return
        # 3. Protocol loop: EHLO, STARTTLS, AUTH, ...
        # 4. Every action → _log_event() with session, TLS status, mode
```

T1 Deep Dive – STARTTLS Capability Stripping (SMTP)

- **Baseline:** Server responds with `250-STARTTLS` → client upgrades to TLS → secure
- **T1 active:** `250-STARTTLS` is missing from the response → client "sees" no TLS option

```python
if u.startswith(b"EHLO") or u.startswith(b"HELO"):
    # T1: STARTTLS is NOT advertised when mode is active
    starttls_advertised = (not tls_active) and (
        (dec.mode not in {"t1"}) or (not dec.active)
    )
    _log_event( ... , event="ehlo", starttls_advertised=starttls_advertised)

    caps = [b"250-selftest", b"250-PIPELINING",
            b"250-SIZE 35882577", b"250-AUTH PLAIN LOGIN"]
    if starttls_advertised:
        caps.append(b"250-STARTTLS")   # ← in T1: NOT added
    caps.append(b"250 HELP")
    io.send(b"\r\n".join(caps) + b"\r\n")
```

Event Logging – Structure & Examples

Example T1 event sequence (vulnerable client = FAIL):

#	Event	TLS	Meaning
1	connect	false	Client connects on port 587
2	ehlo	false	starttls_advertised: false (T1!)
3	auth_command	false	⚠️ Client sends AUTH without TLS
4	disconnect	false	Connection closed

Example T1 event sequence (secure client = INCONCLUSIVE):

#	Event	TLS	Meaning
1	connect	false	Client connects on port 587
2	ehlo	false	starttls_advertised: false (T1!)
3	disconnect	false	Client aborts (no STARTTLS → no login)

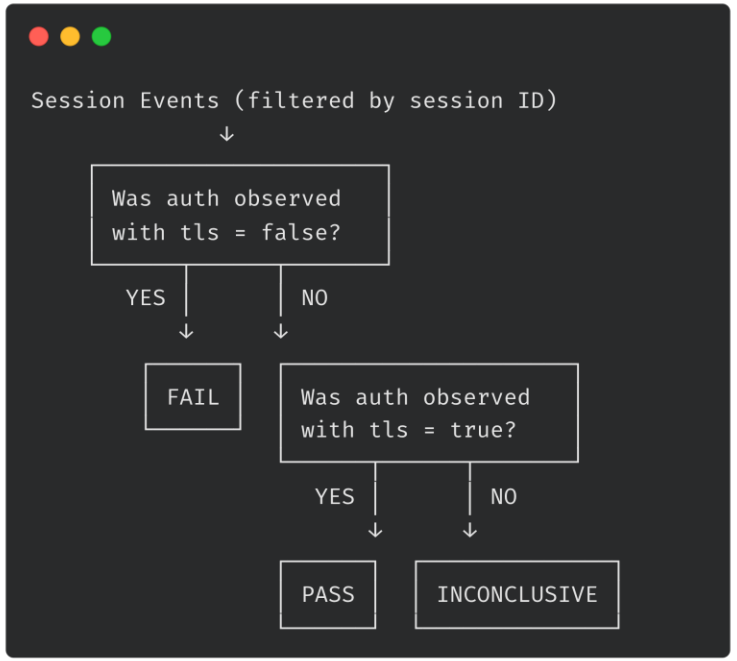

```
{"ts":1738000000,
 "proto":"smtp",
 "client_ip":"203.0.113.42",
 "mode":"t1",
 "mode_source":"override:203.0.113.42",
 "override_session":"a1B2c3",
 "session":null,
 "tls":false,
 "server_port":587,
 "event":"connect"}
```

Possible Verdicts

- **PASS**: Auth/login happened only with TLS (no plaintext credentials seen).
- **FAIL**: Auth/login happened without TLS (plaintext credentials exposure).
- **INCONCLUSIVE**: No auth/login observed (client aborted / got stuck / never reached auth).
- **WARN**: Client showed a security prompt/downgrade warning (user-reported; no plaintext proven).
- **NOT_APPLICABLE**: Test step couldn’t be executed (cannot connect; user-reported).
- **SKIPPED**: Step was skipped in Guided mode

Event Evaluation – Verdict Computation

Signal	Source	Meaning
auth_plain count	Event log (SMTP + IMAP)	Auth attempts observed without TLS
auth_tls count	Event log (SMTP + IMAP)	Auth attempts observed with TLS active
retry_like	≥6 connects, no auth	Client stuck in retry loop (keeps reconnecting)
starttls_refused_like	STARTTLS refused / dropped / wrap failed	Server disrupted TLS upgrade (testcase behavior)
User report: prompt	Manual user input	Client showed a security/downgrade prompt → WARN
User report: cannot_connect	Manual user input	Client could not connect at all → NOT_APPLICABLE



Technical Challenges & Design Decisions

- **Session encoding in username:** `test-SESSION@domain` → server extracts session from username, since no other channel is available (no cookies/headers in SMTP/IMAP)
- **No real MITM required:** The server itself simulates attack behavior → no mitmproxy on client side, no custom CA installation needed
- **Implicit TLS blocking:** In T1–T4, ports 993/465 are immediately disconnected → forces clients onto STARTTLS ports where downgrade behavior is observable
- **Privacy-safe logging:** Passwords are never logged, only username/session + TLS status
- **TTL-based overrides:** Mode store with expiry per IP → no persistent state changes, automatic cleanup
- **Dual storage architecture:** Event log (JSONL) is append-only and crash-safe
- **Persistent historical results:** Guided run data and user-submitted session reports are never pruned

Pre-Authentication Session Binding Problem

Client connects (port 587) ← only IP known

↓

Server sends: 220 ESMTP ← still only IP is known

↓

Client: EHLO ← server responds with capabilities (T1 decision HERE)

↓

Client: AUTH PLAIN <base64> ← session ID only available NOW (too late)

Problems:

- NAT / shared IPs
- CGNAT (mobile carriers)
- VPN / proxy
- Dynamic IP changes
- Race condition

Why no POP3?

- “legacy” protocol
- configuration effort for the user
- Result: adding POP3 would likely reduce completion rate for a public self-test
- For the initial service, IMAP + SMTP submission covers the most common real-world configurations

Existing Tools?

	EAST	CheckTLS	BadSSL	Our Selftest Service
Target	Email client (MUA)	Email server (MTA)	Web browser	Email client (MUA)
Interface	CLI / local VM	Web form	Website	Web app + credentials
Audience	Researchers	Admins	End users	End users / Admins
STARTTLS stripping test	Yes (simulated)	No (passive only)	N/A	Yes (simulated, T1–T4)
Setup required	Local lab environment	None	None	None
Protocols tested	SMTP, IMAP, POP3	SMTP (MTA-to-MTA)	HTTPS	SMTP, IMAP
Key limitation	Requires local proxy + CA	Tests server, not client	Web only, no email	IP-based session binding

	Lynis	testssl.sh	postfix check	Our Server Checker
Focus	Full system audit	TLS handshake analysis	Config syntax	Config logic (auth + TLS)
Depth	Broad (OS, apps, network)	Broad (OS, apps, network)	Superficial	Deep (interdependencies)
Test logic	"Is the service running?"	"Is RC4 enabled?"	"Typo in config?"	"Is auth without TLS possible?"
Mail-specific	Minimal	STARTTLS check only	Postfix only	Postfix + Dovecot
Config source	Static files	Network probes	Static files	Runtime config (postconf -n, doveconf -n)
Cross-check	No	No	No	Yes (e.g., master.cf overrides main.cf)
Output	Compliance report	TLS details	Errors/warnings	Findings + fix recommendations