RP2: Automated end-to-end email component testing

MSc. Security & Network Engineering

Kevin Csuka, Isaac Klop

October 16, 2018

University of Amsterdam
Supervisor: Michiel Leenaars, NLnet
E-mail software is complex

Large surface for human error

How do you know you did it right?

Anxiety around managing own mail server

Misses an automated end-to-end test
• E-mail software is complex
• Large surface for human error
• E-mail software is complex
• Large surface for human error
• How do you know you did it right?
• Anxiety around managing own mail server
• E-mail software is complex
• Large surface for human error
• How do you know you did it right?
• Anxiety around managing own mail server
• Misses an automated end-to-end test
To what extent can we prove a mail server is properly set up via end-to-end component testing?
Related Work

- End-to-end integration testing [Paul, 2001] [1]
Related Work

- End-to-end integration testing [Paul, 2001] [1]
- Internet.nl [2]
- mail-tester.com [3]
- MxToolbox [4]
- emailsecuritycheck.net [5]
Related Work

- End-to-end integration testing [Paul, 2001] [1]

- Internet.nl [2]
- mail-tester.com [3]
- MxToolbox [4]
- emailsecuritycheck.net [5]

- Not end-to-end
- Not automated
Divided in 3 parts:

1. Taxonomy
   • End-to-end testing
   • Black box
   • RFC/Specifications/Best Practices

2. Design tests
   • Modular
   • Continuous Integration / Continuous Deployment (CI/CD)
Divided in 3 parts:

1. Taxonomy
Method

Divided in 3 parts:

1. Taxonomy
2. Design tests
   - End-to-end testing
   - Black box
   - RFC/Specifications/Best Practices
3. Proof of concept
   - Python3
   - Modular
   - Continuous Integration / Continuous Deployment (CI/CD)
Method

Divided in 3 parts:

1. Taxonomy
2. Design tests
   - End-to-end testing
   - Black box
   - RFC/Specifications/Best Practices
3. Proof of concept
   - Python3
   - Modular
   - Continuous Integration / Continuous Deployment (CI/CD)
Figure 1: Taxonomy of the e-mail architecture
• Expected behaviour of components
• Refer to the respective RFC/Specification
Results - Test Design

- Expected behaviour of components
- Refer to the respective RFC/Specification

E.g. SPF [6]
  - HELO domain, MAIL FROM domain, IP address
  - Is IP address authorized for domain?
  - Returns result code (i.e. pass, fail, softfail etc.)
  - RFC guidelines for result
Figure 2: SPF test design example
Proof of Concept

- Multiple mail servers
  - Public IP address
  - Different configuration
  - Intentional flaws in configuration/DNS records
  - Automated via Ansible

<table>
<thead>
<tr>
<th>Components</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAP</td>
<td>✓</td>
</tr>
<tr>
<td>SMTP</td>
<td>✓</td>
</tr>
<tr>
<td>SMTP-AUTH</td>
<td>✓</td>
</tr>
<tr>
<td>TLS</td>
<td>✓</td>
</tr>
<tr>
<td>DANE</td>
<td>✓</td>
</tr>
<tr>
<td>SPF</td>
<td>✓</td>
</tr>
<tr>
<td>DKIM</td>
<td>✓</td>
</tr>
<tr>
<td>DMARC</td>
<td>partial</td>
</tr>
<tr>
<td>SRS</td>
<td>✗</td>
</tr>
<tr>
<td>Greylisting</td>
<td>partial</td>
</tr>
<tr>
<td>Spamfilter</td>
<td>partial</td>
</tr>
<tr>
<td>Sieve</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table 1:** Components which the test suite can and cannot verify
Proof of Concept - Limitations

- Guidance from RFC/specification is limited
  - SPF softfail [6]
  - Greylisting [7]
  - Various errors
- DMARC sending report
- SRS
- Spamfilter
Proof of Concept

Figure 3: Test suite - test run
Conclusion

- Tool assures administrator components work properly
- Limitations
• Not all test cases covered - no complete taxonomy
• Opinionated (RFC often states SHOULD)
• End-to-end testing vs. unit/integration testing
Future Work

- Complete topology/taxonomy of e-mail infrastructure components
- Spam filter
- Expand current tests, e.g. ARC [8], edge cases
- Form of authentication for the test mail-servers
- Comparison study


Figure 4: SPF breaking e-mail forwarding