Lecture Notes for Web Security 2018

Part 5 — E-mail Security

Martin Hell

This chapter will discuss some different security aspects of emails. In order to understand these aspects, it is necessary to understand the SMTP protocol, i.e., how emails are sent from one user to another, and what the messages look like.

1 The SMTP Protocol

SMTP (Simple Mail Transfer Protocol) is the most common protocol used to transfer emails. It was first defined in RFC 821 [6], which was obsoleted by RFC 2821 [4], which was again obsoleted by RFC 5321 [5]. The email architecture consists of several building blocks, see Figure 1.

- MUA - Mail User Agent. This is the source and target of emails and is responsible for delivering the email to the MSA. Typical examples are Eudora, Outlook, Pine and Kmail.
- MSA - Mail Submission Agent. This receives the message from the MUA and delivers it to the MTA. This is usually implemented together with the MTA instead of being a standalone program. The MSA can implement security checks on the email, e.g., checking that it is not spam, before sending it to the MTA. Message submission uses TCP port 587 and is discussed in RFC 4409 [3].
- MTA - Mail Transfer Agent. This is the building block that implements the SMTP protocol. It is responsible for both sending and receiving messages. When the destination has been reached, the MTA at the end host sends the message to the MDA. Examples of MTAs are sendmail, postfix, qmail and Microsoft's Exchange Server.
- MDA - Mail Delivery Agent. The MDA receives the message from the MTA and delivers it to the MUA. Example of MDA are Procmail and maildrop.

It is also common to place an MRA (Mail Retrieval Agent), between the MDA and MUA. However, this is not a standard component in the email architecture.

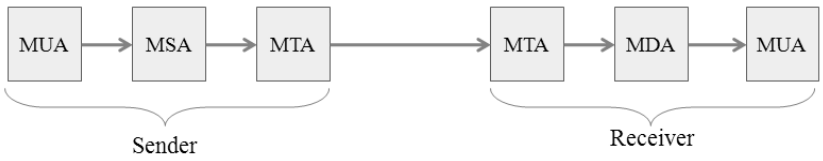

Figure 1: Email architecture overview.

Examples of MRAs are fetchmail and getmail. These clients do not include a user interface. This is left to the MUA. It can be noted that the building blocks are not necessarily implemented separately. It is e.g., common to include the MDA in the MTA implementation.

The SMTP communication uses TCP port 25 and involves two hosts, the SMTP client and the SMTP server. The server is the host receiving an email and the client is the host that sends the email. Both the client and the server functionality is implemented by the MTA, and one MTA can act as both server and client for one message if it is relayed through the host. In the original specification, RFC 821, the terminology smtp-sender and smtp-receiver was used instead of client and server.

1.1 Commands and Replies

When the TCP handshake is completed the client sends a sequence of commands to the server in order to define recipients, the sender and to transfer the actual message. A typical SMTP communication consists of the following steps.

- HELO and EHLO - A hello message that initiates the communication. EHLO means extended HELO and should be sent if the client supports SMTP extensions. Together with this command the client host also identifies itself by sending its hostname.
- MAIL - This command is used to identify the sender of the email. This information is given as `MAIL FROM:sender@client.com`. This address is also known as the *envelope sender address*. If the sender is accepted the server replies with a `250 OK` reply.
- RCPT - This command is used to specify the receiver(s) of the email. It is given as `RCPT TO:receiver@server.com`. The command is used once for each receiver. If the receiver is accepted the server responds with a `250 OK` reply.
- DATA - This command tells the server that the following input is the *message content*. This is also referred to as the *mail data*. Upon receiving the DATA command the server replies with `354 Intermediate reply`, which tells the client that the following lines will be treated as mail data.

To indicate the end of mail data the client sends a period "." on a single line. A `250 OK` is sent by the server if the mail data submission was accepted. Other commands include VRFY, which asks the server to verify an address, EXPN which asks the server to expand a mailing list, QUIT which ends the communication, RSET which resets the current transaction and HELP which asks the server to return help of commands. VRFY can be used by spammers to verify that email addresses exist and EXPN can be used by spammers to harvest email addresses from mailing lists. For this reason, these commands can be disabled by the server.

The commands and the command arguments are not case sensitive. They can be written in lower- or uppercase letters, or even as a mixture between lower- and uppercase. The only exception to this rule is that the local part of the email address must be treated as case sensitive. The local part is the name given to the left of "@" character. The reason is that the end host itself might make a distinction between lower- and uppercase letters, though this is discouraged. (The fact that commands and arguments are case insensitive is stated in the standard, but this does not necessarily mean that all implementations follow the standard.)

Replies sent by the server are used to indicate to the client to which extent the command was successful. A reply is a 3-digit number and there are 4 main categories, starting with digits 2, 3, 4 and 5 respectively. A reply of the form `2XX` is called a *positive completion reply* and indicates that the command was successful and the client can continue with a new command. A reply of the form `3XX` is a *positive intermediate reply*. It means that the command was successful, but that the client is not done with the command yet. The `354` Intermediate reply" that is sent upon receiving the DATA command is an example. The client is not done until a period is sent as the only character on a line. A *transient negative completion reply* is of the form `4XX` and indicates that the command was not successful. However, the client is encouraged to try the command again at a later time. It might be successful then. By contrast, a reply of the form `5XX` is a *permanent negative completion reply* and means not only that the command was not successful, but it also means that the client should not try it again. There is no chance it will ever be accepted. An example of an SMTP communication is given below. "S" denotes server messages and "C" denotes client messages.

```
S: 220 server.com Ready
C: EHLO client.com
S: 250-server.com greets client.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<sender@client.com>
S: 250 OK
C: RCPT TO:<rec1@server.com>
S: 250 OK
```

```
C: RCPT TO:<rec2@server.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: This is my message
C: .
S: 250 OK
C: QUIT
S: 221 server.com Service closing transmission channel
```

1.2 The Email Message

The message consists of two parts, the message header and the message body. Both are sent using the DATA command and are separated by an empty line. The header includes information about the email and is often hidden from the end user by the MUA. However, MUAs usually have functionality that allows the user to easily see the header. There are several different headers that are common. The `To` and `From` headers are provided by the sender. They give information about who the email is sent to and from. The address given in the `From` header is the *header sender address*. This address is used by the email program to display the sender to the receiver. The SMTP server usually does not care about this address. Since they are provided by the sender of the message these are very easy to forge. There does not have to be any link between the information in the `From` header and the address given as an argument to the MAIL FROM command.

Another header is the `Return-path`. Other names for this header are bounce address, return path and envelope from. This header is added by the last SMTP server and it gives the argument of the MAIL FROM command. It can be used for e.g., error messages if the destination mailbox does not exist.

A `Message-id` header is added by the first SMTP server and a `Received` header is added by each SMTP server that handles the message. New `Received` headers are added before older ones. These headers can be used to analyze the path taken by a particular message. An example of a `Received` header is given below.

```
Received: from mail.sender.com (mail.sender.com [123.45.67.89])
by mail.receiver.com with ESMTP id 31si3889671fkt;
Fri, 03 Oct 2008 00:49:45 -0700 (PDT)
```

This header is added by the server `mail.receiver.com`. The `mail.sender.com` is the name the client identified itself as in the HELO/EHLO command. Then the IP address used in the TCP connection together with the reverse DNS lookup of this IP address follows. In this case the server has a TCP connection with a host with IP 123.45.67.89. A reverse DNS lookup is used to find out that the name of this computer is `mail.sender.com`. The server also adds its own

name together with its own id for this message. The last part is the date and time that the message was received.

The received headers can be used to detect if a message is forged. It is possible to detect if the SMTP client is not who it claims to be. An example of a header of a forged message is given below.

```
Return-Path: [fake@anywhere.com]
Received: from smtp.server1.com (smtp.server1.com [134.72.98.54])
by smtp.server2.com with ESMTP id 73659812;
Fri, 12 Dec 2007 13:46:54 -0400 (EDT)
Received: from google.com (dklku64.someISP.com [234.56.67.78])
 by smtp.server1.com; Fri, 12 Dec 2007 10:45:28 -0700 (PDT)
Date: Fri, 12 Dec 2007 10:45:28 -0700 (PDT)
From: cheap products <cheap@gmail.com>
To: something@somewhere.org
Subject: The best offer only for you
```

In the bottom received header it can be seen that the client claims to be `google.com`. However, the server has a TCP connection with a computer that has the IP address 234.56.67.78, which turns out to be a computer with host name `dklku64.someISP.com`. This does not match. The reason that it is possible to detect a forged email like this is that it is very difficult to forge the IP address used in a TCP connection.

2 MX Records

The MX-record is a DNS record that specifies to which computer an email to a certain domain should be sent. This computer does not necessarily have to be the final delivery server that delivers the email to the MUA. It can be a firewall, a server shared by several domains used to filter e.g., spam, or it could be a gateway that forwards the email on to another protocol.

The MTA that is acting as client in the SMTP communication looks at the recipient domain and queries the DNS corresponding to that domain for the IP of the SMTP server that the client should connect to. After getting a reply, the SMTP client connects to the SMTP server and delivers the email.

It is possible to list several servers in the MX record. Each server has a corresponding priority number where lower number means higher priority. The server with the lowest number should always be used first. In case that server is not available, i.e., it is offline or for some other reason does not respond, the server with second priority should be used. Servers that do not have the highest priority are backup servers. If the main SMTP server is offline, and there are no backup servers, the SMTP client will queue the message and try resending it every now and then. The problem in this case is that the client does not know when the main server is online so it might try resending the message several times before it succeeds. If backup servers are used the client can send the email to a backup server, which can then deliver the email to the main server

as soon as it is online again. The main server can e.g., send a notification to the backup server when it is online again.

The priority number can also be used for load balancing. This is achieved by giving several servers the same priority. In that case the client must randomly pick one of the servers with lowest priority.

3 Open Relays

An open relay is an SMTP server which can be used by anyone to send emails to anyone. The relay does not verify that the sending or receiving machine is allowed to use the server for SMTP communication. These open relays are popular among spammers. Historically, open relays were very common but as spam became more and more widespread, the number of open relays has decreased. Today, most SMTP servers are closed. This means that they only accept or receive emails in which either the sender or the receiver is considered local by the server.

4 Authenticating E-mails and Senders

4.1 DKIM

DKIM is an abbreviation of DomainKeys Identified Mail. The standard is given in RFC 6376 [1]. DKIM adds a digital signature to an email message. The signature applies to the message body and a subset of the headers in the email. Any header not included in the signature can be changed or deleted during transit without changing the validity of the signature. Thus, it is desirable to include as much as possible in the signature. The signature is normally created by (one of) the first SMTP server(s). As a result, the signature can not include `Received` headers added by receiving SMTP servers. Exactly which headers are included in the signature is defined and given as a part of the signature. An example of a DKIM signature is

```
DKIM-Signature:
v=1;
a=rsa-sha256;
c=relaxed/relaxed;
d=gmail.com;
s=gamma;
h=domainkey-signature:received:received:message-id:date:from:
to:subject:mime-version:content-type;
bh=9gicsZnlcLK7yYh6VIrgyAMMRZiWsSbWqSPIhc78RRk=;
b=k4ofvpHPkaQmvuSoGVhRrnCsPK+JEuv9KUrZO7aiypvf/6Y1N2iIatvLvdzwOn
ZX/W6Kxyx6Z4Ybuk8Dqk/vNTIE7Jpy+GQUUHFvM0NFtmZo1CbGRvo8DdHnXRBB/q
WwlV+Z6wxw/mq7lNuJknVprOAaTLws5mwcZ+AWL8KwHg0=
```

The DKIM signature header consists of a set of tags and values in the form "tag=value". A few of them are explained below. Consult the specification for a complete treatment of all tags.

- **Version number (v):** The version number of DKIM. It is currently 1.
- **Algorithm (a):** The algorithm used in the signature.
- **Signed header fields (h):** A colon separated list of which header fields that are included in the signature. The DKIM signature header field is not included in this list, but is still always included in the signature.
- **Domain (d):** The domain signing the message.
- **Selector field (s):** A domain can have several public/private key pairs used to generate signatures. To separate them, a selector is used. Thus, the signature is identified by both the domain and the selector. Any party receiving and/or forwarding the message can verify the signature, but the typical case is that it is verified by the end recipient or an agent in the recipient's domain. Exactly which agent is responsible for the signing and verification of the message is determined by the signer/verifier. Thus, it can be either the MUA, MSA/MDA or the MTA that implements the signing/verification of the signature.
- **Canonicalization field (c):** Some SMTP servers make small changes to the header and/or body of the message. This may or may not be allowed by the signer. This field defines if the canonicalization is simple or relaxed. Simple canonicalization means that no changes are allowed in the header and ignores only extra newlines at the end of the body. Relaxed canonicalization means that even if certain changes are made to header and/or body the signature remains valid. This is achieved by signing a normalized version of the message. As an example, header name fields are always treated as lowercase. When the message is received, it is converted into this normalized form and this version of the message is verified.
- **Body hash (bh):** This is the hash value of the canonicalized message body, encoded with base64.
- **Signature data (b):** The actual signature, which is encoded using base64.

The DKIM signature header field should always be included in the signed part of the message. Obviously, the signature itself has to be excluded. This is solved by treating the value of the b-tag as an empty string. When a message is signed, the header fields defined by the h-tag are hashed and this value is signed. Since the hash of the body part of the message is included in the DKIM signature header, the message body is also signed. A message can have more than one signature.

4.1.1 Verifying the Public Key

DKIM allows a domain to take responsibility for the integrity of an email. This can be compared to the case with PGP and S/MIME in which the message is signed by the author of the email. The domain level granularity of DKIM does not require that each user has a public/private key pair. Instead, only the domain is required to have a key pair. One of the main problems in public key cryptography is the need to verify the connection between an identity and a public key. The common solution is to use digital certificates. This is the approach in e.g., SSL where the client (and possibly also the server) needs to have predistributed root CA certificates in order to verify the public key in the server (or client) certificate. PGP takes a similar approach in its web of trust. Users can sign each other's certificates with the idea that if your friends trust a certain user, then you implicitly also trust that user. DKIM has taken a different approach, namely that the public key is located in a DNS entry for the domain. This avoids the problem of root CA certificates and is made possible by the fact that it is the domain, not the user, that signs the email. A dedicated subdomain, called *_domainkey*, is used for the DKIM public key. Thus, when receiving an email signed with a DKIM signature, the verifier contacts the DNS of the signing domain to get the public key used to verify the signature. The idea behind this is that only administrators of the domain itself are allowed to change DNS entries of that domain. We can say that the security of the public key authenticity is moved from the PKI to the protection of the DNS, which can be solved by symmetric cryptography and user authentication. Below is an example of a DNS query asking for the public key for `gmail.com` with selector `gamma`.

```
>nslookup -type=txt gamma._domainkey.gmail.com
Server:      ***
Address:     ***

Non-authoritative answer:
gamma._domainkey.gmail.com      text = "k=rsa; t=y;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDIhyR3oItOy22ZOaBrIVe9m/
iME3RqOJeasANSpg2YTHTYV+Xtp4xwf5gTjCmHQEMOs0qYuOFYiNQPQogJ2t0Mfx
9zNuO6rfRBDjiIU9tpx2T+NGlWZ8qhbiLo5By8apJavLyqTLavyPSrvsx0B3YzC6
3T4Age2CDqZYA+OwSMWQIDAQAB"
```

The p tag is the base64 encoded public key and the k tag determines what type of key it is. The default is `rsa` which means that it is an ASN.1 DER-encoded RSA public key. This is the only key type that must be supported. The tag t=y means that the domain is testing DKIM, but the signature must be used as if it was not testing. The only difference is that the verifier can help the signer by providing test results.

While DNS is the initial and current method of fetching public keys, there is nothing in the DKIM design that requires DNS to be the public key repository.

If other, more suitable, services become available, DKIM can be extended to use these as well.

4.1.2 Algorithms

In DKIM, RSA should be used as signature algorithm and SHA-256 should be used as hash algorithm when signing. The verifying part must also implement SHA-1 in case this is used as hash function. The main reason for this is that SHA-1 is significantly faster than SHA-256 and some senders may wish to use this faster algorithm instead, despite its (theoretical) security problems. The set of algorithms might be extended in the future.

4.2 SPF Records

The HELO/EHLO command specifies the computer connecting the SMTP server and the MAIL FROM command specifies who is sending the email. As noted before, the domain given in the HELO/EHLO command can be arbitrarily chosen by the connecting computer. Furthermore, similar to the sender address on an envelope, the address given as argument to the MAIL FROM command can be arbitrarily chosen by the sender of the message. Thus it is possible to connect to an SMTP server and send an email from any email account. The Sender Policy Framework (SPF), defined in RFC 4408 [7] is a way of determining who is allowed to send emails from a certain domain. This is in the interest of both the domain owner and the recipient. As a domain owner of `company.com` you do not want to give people, maybe competing companies, the possibility to send emails from an email address at `company.com`. Not only can this possibility be used to discredit the company and spread false information, it can also be used to fill up the inbox of users at the company with error and out-of-office messages. As an example, assume that Mallory sends 1.000.000 emails to different accounts. If 0.1% of these are sent to non-existing accounts or accounts which have an automatic out-of-office reply, the claimed sender will get 1000 emails. This is known as *backscatter*. Also the recipient has an interest in knowing that an email from `company.com` actually comes from a computer at `company.com` and is e.g., not a phishing attack.

For SPF to work, both the domain administrator and the server receiving the email have to implement their respective part. The domain administrator has to specify which computers that are allowed to send an email from that domain. Moreover, when a connection is made to an SMTP server, that server has to verify that the computer initiating the connection is allowed to send an email from the domain given in the MAIL FROM command. The specification recommends that the domain given in the HELO/EHLO command is verified, while the domain given in the MAIL FROM command must be verified.

The list of IP addresses allowed to send email from a domain has to be kept secure so that an adversary can not insert arbitrary hosts. At the same time the list must be available at any time for SMTP servers to read. Similar to DKIM, this is accomplished by putting this information in the DNS that is

authoritative for the domain. Comparing the SPF record with MX records in the DNS, it can be noted that they accomplish the reverse of each other. While the MX record specifies which computer that receives email designated to the domain, the SPF record specifies which computers (are allowed to) send email from the domain.

The SPF record is a distinct DNS record type, but it is identical to TXT resource records. Since not all DNS implementations support the SPF record type, both SPF and TXT record types should be used in servers. However, only the TXT type will be used in the examples here.

The text strings used to define if a computer is allowed to send the email can be quite complex. A detailed overview is given in RFC 4408. When the SPF client (receiving SMTP server) performs a check, it takes the domain given in the MAIL FROM and the IP address of the sending SMTP server, which is given in the TCP connection. By looking at the corresponding SPF or TXT record for the domain, it is decided if the IP is allowed to send the email. Some *mechanisms* used here are:

- **all**: Any IP address will match this.
- **a**: The IP address will match if the domain has an A record that resolves to the sender's IP address.
- **mx**: The IP address will match if the domain has an MX record that resolves to the senders IP address.
- **ipv4**: The IP address matches an address in the given IPv4 range. There is also an IPv6 variant with the same meaning.

One of four qualifiers is used to decide what to do when an IP address match:

- `+`: Check result is "pass". This is the default if no qualifier is given.
- `?`: Check result is "neutral".
- $\sim$: Check result is "softfail". This is typically used for testing and debugging and can be used to tag a message as suspicious.
- `-`: Check result is "fail". The message can be rejected.

Using the mechanisms and qualifiers, a simple SPF record is given below.

```
example.com.   SPF   "v=spf1 a mx -all"
```

The first part of the string (`v=spf1`) specifies the version. Then follows three directives. Assume that the sending SMTP server claims that the email comes from `alice@example.com` (in the MAIL FROM command). First, the sender passes the SPF check if example.com has an A record that resolves to the IP address used. Otherwise, it passes if there is an MX record that resolves to the IP address. Note that the mail server used for example.com can be in a completely separate domain, such as server.net. This situation is solved by

using the MX mechanism. If neither of these two returns a pass, the check will fail since the IP address will always match the "all" mechanism. If there is no match, "neutral" is the default result. The SPF client will thus make (at most) the following DNS queries:

1. Query the SPF record for example.com.
2. Query the TXT record(s) for example.com.
3. Query the A record(s) for example.com.
4. Query the MX record(s) for example.com.
5. Query the A record(s) for the host(s) returned from the previous query.

Thus, it is clear that an SPF check can generate quite many DNS queries. For this reason there is a limit of at most 10 DNS queries not counting the queries used to get the actual SPF information. The following is another example of an SPF record:

```
example.com.   SPF   "v=spf1 a:sub.example.com  ipv4:1.2.3.4/24"
```

In this case the first directive specifies that the check passes if an A record in sub.example.com resolves to the IP address of the sending SMTP server. It also passes if the IP address is in the range 1.2.3.0 - 1.2.3.255. Otherwise the check results in a neutral result.

In the treatment above, only the domain part has been considered. However, it is possible to have rules that depend on the local part of the address given in the MAIL FROM command also. Refer to the specification for more info on this.

SPF is not primarily a tool for fighting spam. If a spammer wants to forge a sending email address, he can just use an address in a domain that does not support SPF. It must also be noted that SPF checks the address in the MAIL FROM command. It does not check what is written in the `From` header, which is actually the one used by email clients to indicate the sender of the email.

4.3 DMARC

Both DKIM and SPF are used to authenticate email messages, but they do it in different ways. A limitation is that once the sender has implemented DKIM and/or SPF, it does not know what effect or consequences it has. In the case of DKIM, the sender does not know if there are many emails from the domain that have a bad signature. In the case of SPF, the sender does not know if there is a mistake in the list of allowed IP addresses so that one computer will always fail the SPF check at the receiver side. Another limitation is that the sender does not have any control over what should happen if there is a problem with the DKIM signature and the SPF check fails. One receiver might automatically treat the message as spam while another might let it through without any action. Domain-based Message Authentication, Reporting and Conformance (DMARC)

is an effort to standardize a way for senders to announce that they are using DKIM and SPF, to let the sender recommend an action to take if the checks fail, and to let receivers give feedback to the sender. The specification is given in [2] and the reader should refer to this document for details.

DKIM and SPF uses different identifiers. While DKIM uses the "d=" tag in the signature, SPF uses the domain given in the MAIL FROM command in SMTP. DMARC has chosen to tie together the identifiers by using the domain given in the "From" header of the message. This header is most often the one used by MUAs and shown to the users. Messages are said to be *in alignment* if the different identifiers have the same domain. In *strict mode* the domains have to be identical, while in *relaxed mode* it is enough that the organizational domain is the same. The organizational domain is basically a TLD plus one more label, e.g., example.com or server.net etc. As an example, assume that the "From" header is `Alice@home.example.com`, the value of the "d=" tag in the DKIM signature is `example.com` and the MAIL FROM domain identifier is `home.example.com`. Then the DKIM identifier is in alignment in relaxed mode, but not in strict mode, while the SPF identifier is in alignment in both strict and relaxed mode.

The sender has three options for which policy a receiver should apply to messages that fail authentication.

- **None.** No action should be taken, but the result of the checks should be sent as feedback to the sender. This is called monitor mode.
- **Quarantine.** The receiver should treat the email as suspicious. Exactly what that means is up to the receiver, but a typical action could be to mark it as spam.
- **Reject.** The receiver should completely reject the message without delivering it to the user.

The intention is that the deployment of DMARC can be incremental, starting with "none" in the beginning for testing purposes. Feedback from receivers can be used to make sure that all legitimate traffic is properly protected. Then the action is shifted to "quarantine" and when the sender is certain that everything works as intended, the action is shifted to "reject". In order to soft shift between the actions, a percentage is also used. Quarantine with 50% means that half the messages received should be treated as if the action was "quarantine", while the rest should be treated as if the action was "none". Using "reject" with e.g., 20% means that 20% should be rejected, while the rest should be quarantined.

The feedback allows a sending domain to receive information about its DMARC deployment. This information can be very valuable and is an important feature in DMARC. It allows the sender to get information about attempted attacks and perhaps problems with its own infrastructure that would otherwise not be possible to get. A receiver collects data about emails from a particular domain and sends feedback reports typically once per day. It should include both information about messages that passed the authentication and those that

failed. The information should include e.g., identifier alignment, SPF and DKIM results, the policy that was requested and the policy that was actually applied by the receiver. Receivers are only required to send this report daily, but can send it more often if the sender requests it. Furthermore, receivers only have to support sending the reports by email, but any URI can be requested by the sender. A common alternative is to provide a HTTP URL to which the report is sent in a POST message. In addition to the aggregate reports, message specific forensics reports for messages that fail authentication can also be sent.

A DMARC record uses the "tag=value" syntax to define the sender's options. Some of the tags that are possible are given below.

- **v**: This tag is required and currently only "DMARC1" is possible.
- **p**: This is the policy that the sender wants the receiver to use (none, quarantine or reject). This tag is required.
- **pct**: The percentage (0-100) of the messages to which the policy is applied. Default is 100.
- **rua**: URI to send aggregate feedback to.
- **ruf**: URI to send forensics information for specific messages to.
- **adkim**: Determines if strict or relaxed alignment mode should be used for the DKIM identifier. Default is "r" (relaxed), strict mode can be chosen using the string "s".
- **aspf**: Determines alignment mode for the SPF identifier similar to adkim.
- **sp**: Same as the p-tag, but the policy applies to subdomains of the queried domain.
- **ri**: Time interval in seconds for the aggregate reports. Default is 86400 which corresponds 24 hours, i.e., once per day.
- **rf**: Format to use for the forensic information.

Similar to DKIM and SPF, DMARC takes advantage of the integrity protection given by the DNS to store information about a domain. The DMARC record is given in a DNS TXT record using `_dmarc` as prefix for the domain. An example of a DMARC record is given below.

```
>nslookup -type=txt _dmarc.paypal.com
Server:     ***
Address:    ***

Non-authoritative answer:
_dmarc.paypal.com       text = "v=DMARC1; p=reject;
rua=mailto:DL-PP-DK-Reports@ebay.com;
ruf=mailto:dk@bounce.paypal.com;"
```

This example gives DMARC info for paypal. It provides the version and spec-

ifies that messages that fail DKIM and SPF checks should be rejected (100% of the time). It also provides two email addresses for receiving feedback and forensics information. Note that a domain different from paypal is used to receive feedback. This is permitted, but also allows for a malicious domain to direct unwanted reports to a victim domain. However, this is not considered a serious problem partly because this feedback is not sent often enough to allow a practical DOS attack and partly because the malicious domain is clearly visible so appropriate actions can be taken against it if necessary.

Several huge email senders support DMARC including Gmail, PayPal, Ebay, AOL, Hotmail, Facebook and LinkedIn to name a few. A widespread adoption has the potential of efficiently preventing many phishing attacks where the attacker spoofs the sender identification.

5 Preventing and Detecting Spam

Spam is typically defined as unsolicited email that is sent in bulk to many receivers at the same time. Spam makes up a significant portion of all emails that are sent. Estimated figures are sometimes around 90% of all email messages, but an exact figure is of course difficult to find. Unwanted email are not only irritating, but also time, space and bandwidth consuming. Several techniques for preventing and detecting spam have been proposed and some will be discussed in this section.

While DMARC can be used to combat certain powerful phishing attacks, such as forged emails, it can not detect spam messages in general. Only if the spam includes some information leading to an authentication failure, the spam would be detected. However, the spammer might not want to e.g., disguise as another company. In fact, a spammer is free to use DMARC to authenticate the spam email, though this would of course make the spammer much easier to trace. To prevent and detect spam, other methods are needed that take into consideration the different ways of generating spam messages.

5.1 DNS Blacklist

Some MTAs are used by spammers more often than others. A DNS blacklist (DNSBL), or DNS blocklist, is simply a list of IP addresses that are known to having been used by spammers. Another name for these lists is Realtime Blacklist (RBL). The IP addresses can e.g., come from a honeypot, usually referred to as *spamtrap* in this context. The MTA acting as a server can, upon a connection made by a client MTA, check if the client IP is listed on the blacklist. If it is, the server MTA can refuse to deliver the emails or just simply close the connection. Another option is to combine the result from the blacklist with other properties of the email transaction in order to make a suitable decision whether it is spam or not.

A blacklist is contacted using DNS queries. The query for an IP address at provider `server.com` is constructed as

```
IP(r).server.com
```

where IP(r) is the reverse byte ordering of the IP address. There are several providers of blacklists and it is up to the blacklist provider to determine if an MTA should be put on its list or not. There are some problems that need to be considered in the context of blacklists. First, how do we decide what makes an MTA bad enough to be put on the list? Should we automatically add all open relays even if they are not known to have sent spam before? Should we add an MTA that is known to have sent spam only one time? Depending on how these and related questions are answered, the provider may or may not regard an MTA as qualified for the blacklist. Thus, IP addresses that are considered spam MTAs in one blacklist can be considered legitimate in another blacklist. Some blacklists are known to be more conservative than others. The result of this is that there will always be false positives and/or false negatives. A very aggressive blacklist will add many MTAs to its blacklist. This means that it is very likely that an MTA attempting to send spam is on the blacklist, reducing the number of false negatives. On the other hand, there are likely to be IP addresses that maybe was once used to send spam, but are most often used for legitimate emails. Thus, the number of false positives will increase. The opposite holds for a conservative blacklist that only adds MTAs to the list if it considers it very likely that an email sent from this MTA will be spam. This increases the number of false negatives while decreasing the number of false positives. Below is an example of this situation.

```
>nslookup 253.225.237.209.zen.spamhaus.org
Server:      ***
Address:     ***

**Server cant find 253.225.237.209.zen.spamhaus.org: NXDOMAIN

>nslookup 253.225.237.209.spam.dnsbl.sorbs.net
Server:      ***
Address:     ***

Non-authoritative answer:
Name: 253.225.237.209.spam.dnsbl.sorbs.net
Address: 127.0.0.6
```

Two blacklists are queried for the IP 209.237.225.253. The first does not have it on the blacklist, while the second does. (The address returned is in this case a return code, specifying in which database the IP was found.)

On the other hand, the ambiguity of determining if an MTA should be on the list can be seen as a feature. It it possible for an administrator to decide if an aggressive or conservative list should be used based on which is most appropriate in his/her case, taking false positives and false negatives into account. From a legitimate MTA's point of view, a problem with these blacklists is that if someone uses your MTA to send spam once, maybe because of a

small administrative mistake, that MTA can be blacklisted even though the administrator of the MTA never intended it to be used for spam. This can be seen as a special case of a somewhat larger problem with DNS blacklists, namely that the blacklist administrator has a large responsibility for the included IP addresses. A widely used blacklist has large influence over the IP addresses that can be used to send emails and errors resulting in false positives can have impact on the business for many companies.

5.1.1 URI DNS blacklists

A URI DNS blacklist (URI DNSBL) is very similar to a DNSBL. However, instead of blacklisting IP addresses of MTA, URLs used in the email body is blacklisted. Often spam emails include a webpage to which one can go to order certain items. These URLs can then be blacklisted in a similar way as the MTA IP addresses. The server MTA then looks for URLs in the body of a message and queries a blocklist for these URLs.

5.2 Greylisting

Greylisting is an anti-spam technique that relies on the fact that many spam programs do not fully comply with the SMTP standard. More specifically, in the SMTP standard (RFC 5321) it says that

> *"...mail that cannot be transmitted immediately MUST be queued and periodically retried by the sender".*

The mandatory retry is the idea behind greylisting. When an email is sent by a client MTA, the server looks at the vector

```
(Client IP, sender address, receiver address)
```

and if this vector has not been used before (or perhaps recently), then the email is automatically rejected using a Transient Negative Completion reply. At the same time the vector is added to a database of accepted vectors. The reply informs the client that it was not possible to send the message, but that the client should try again later. If the client complies with the standard, then the email will be sent again later and it will be accepted as the vector will be found in the database.

This is a very simple method to filter spam emails. It does not require interaction with other servers, as in the case with DNSBL, and it is not very resource consuming. It can also be used before other spam filters are activated, e.g., statistical filtering, in order to reduce their workload. However, it also has some drawbacks. Emails will no longer be delivered immediately in the case the vector is not found in the database. According to RFC 5321, the retry interval should be at least 30 minutes, but this is configurable in implementations and it is common to have a shorter interval. This will delay an email without possibly either of the end users being aware of the reason or the fact that it is delayed. The technique also heavily relies on legitimate servers to comply with the standard and implementing the reply functionality.

5.3 Nolisting and Related Techniques

Similar to greylisting, nolisting is an idea that is based on the fact that many spam programs do not comply with the standard. A domain can be served by several SMTP servers, with backup servers having a lower priority. The highest priority server (lowest priority number) should be used first, and if this is not reachable the next server should be used and so on. One possible spam strategy could be to only try to connect to the first server. If this is not reachable, do not waste time by testing the other servers but continue instead to another domain and try to deliver emails there instead. The idea behind nolisting is that (at least some) spammers use this strategy. In nolisting, the highest priority mail server is a dummy server, i.e., a host that does not accept any connections to port 25. Spam software that only tries to connect to the highest priority server will not be able to send emails. On the other hand, if the MTA correctly implements the SMTP protocol it will, after not reaching the first server, try the second one and consequently succeed to send the email. Nolisting is of course only successful to some extent. By correctly implementing the SMTP protocol in spam software this anti-spam technique is completely bypassed. However, even if just a small fraction of the spam emails are blocked by this, it is a very simple technique that may ease the burden of other spam filters.

An ad-hoc method a spammer can use to avoid nolisting is to ignore the highest priority mail server and immediately try to connect to a backup server. This has the additional advantage that a backup server might use less restrictive spam filtering. Another technique, denoted *Unlisting*, can be used together with nolisting. In unlisting, requests to the backup server are denied unless a request to the primary mail server has been made first. Thus, requests to the primary mail server are blocked, but the IP address of the client is saved. If the same client (IP address) makes a request to the backup mail server within some predetermined time period, access is allowed. One problem with unlisting is that mail clients are not required to make the connection to the backup mail server from the same IP address. Thus, also clients that correctly implement the SMTP protocol can be rejected. For this reason, unlisting is not recommended.

A related technique assumes that the spammer, if there are several mail servers, will only try the mail server with lowest priority. Since this mail server is rarely used it might be an old computer where the spam filtering is not updated or even functional. Thus, the strategy is to let the lowest priority mail server be a dummy server. Combining this with nolisting a mail server configuration in the DNS could be as follows.

```
server.com.  MX 10 dummy.server.com
             MX 20 real.server.com
             MX 20 real2.server.com
             MX 30 dummy2.server.com
```

5.4 Hashcash

The idea of paying for sending email has been around for a long time. If the charge for sending one email is very small, this will not be very expensive for normal users but very expensive for spammers that send millions of emails. While this logic might be correct, keeping the service free is regarded as having higher priority. Also, it would be very difficult to manage a system where email senders would pay to send an email, given the architecture of the Internet. In the end, SMTP only specifies how to send bytes from one computer to another over TCP. The same thing HTTP does. On an encrypted link it would not even be possible to tell the two protocols apart.

Hashcash is a way of paying to send emails, but the currency is not money. Instead the sender pays with clock cycles on his computer. If it takes 1ms to prepare an email to send, then 1000 emails can be sent every second. If it instead takes 4s to send an email, it would take more than one hour to send 1000 emails. A user sending $10-20$ emails per day will hardly notice this delay, but a spammer would run into serious problems. This idea is comparable to the idea of using a very slow hash function when hashing passwords. Testing one password is still sufficiently fast, but a brute force will be very slow.

Hashcash is asymmetric in the sense that even if it takes long time to prepare the message, verifying that the message took a long time to prepare is very efficient. To send a message using hashcash the sender must include a string in the header. The string is given by

```
ver:bits:date:resource:[ext]:rand:counter
```

where `ver` is the version number (currently 1), `bits` is a number of bits which indicates how costly the hash value was to compute, `date` is the current date, `resource` is the email address of the recipient, `ext` specifies extensions, `rand` is a random number and `counter` is a counter value. The counter value is first set to one. Then, the string is hashed until the first `bits` bits of the hash value are zeros, incrementing the counter for each failed attempt. If the hash function is secure, it will not be possible to construct a string such that the first `bits` bits are zero in any other way than trying on average 2^{bits} different input strings. When a valid hash value is found, the string is added to the message header. The recipient can immediately verify the hash value by just hashing the string. If the hash satisfy the requirement, the sender must have computed about 2^{bits} hash values prior to sending the mail.

The values given in the string guarantees that the string can not be reused. The date prevents the string to be used in another email to the same recipient provided that the recipient remembers the strings that were received during a few days back (if some error in the date is accepted). The recipient address prevents the string to be used for the same email but to another recipient. The random value is used to separate different senders. Otherwise two senders can generate the same string and the last one received will be rejected since it has already been used.

Hashcash can not be used to reject emails as spam since it would require that all implementations support and use hashcash. Instead, it can be used to verify that an email is not likely to be spam. It will not block an email from a legitimate sender, which is e.g., possible when blacklists are used. A drawback is that spammers often control many computers in a botnet. These computers can be used to compute valid hash values. In that case the spammers are not paying, but innocent and unknowing users are providing the payment, i.e., CPU time. To make it difficult to use botnets, the value of `bits` can be increased. Then it would not even be enough to have many computers, it would still take too much time to send enough emails to support the business. On the other hand, at some point the amount of computation will be noticeable also for legitimate users, sending only few emails. In particular if the computer is a few years old.

5.5 Statistical Filtering

Spam emails often have specific characteristics that are not found in legitimate emails. Statistical methods can be used to look for these characteristics and determine if an email is likely to be spam or not. Bayesian spam filtering is the standard method implemented in many spam filters. Training data is used to teach an algorithm which emails are spam and which are not spam. With enough training data, the algorithm can take a message and determine if it is spam or not. The main ideas can be described as follows. Let D be the event that a message has a certain set of words, denoted $w_0, w_1, w_2, \ldots$. Let S be the event that a message is spam and S' the event that a message is not spam. If we assume that words appear independently, we can write

$$\begin{aligned} \Pr(D|S) &= \prod_i \Pr(w_i|S) \\ \Pr(D|S') &= \prod_i \Pr(w_i|S') \end{aligned}$$

By using Bayes' theorem $\Pr(S)\Pr(D|S) = \Pr(D)\Pr(S|D)$ we can write

$$\begin{aligned} \Pr(S|D) &= \frac{\Pr(S)}{\Pr(D)} \prod_i \Pr(w_i|S) \\ \Pr(S'|D) &= \frac{\Pr(S')}{\Pr(D)} \prod_i \Pr(w_i|S') \end{aligned}$$

Dividing $\Pr(S|D)$ by $\Pr(S'|D)$ gives

$$\frac{\Pr(S|D)}{\Pr(S'|D)} = \frac{\Pr(S)}{\Pr(S')} \prod_i \frac{\Pr(w_i|S)}{\Pr(w_i|S')}$$

By taking the logarithm of both sides, we get

$$\log \frac{\Pr(S|D)}{\Pr(S'|D)} = \log \frac{\Pr(S)}{\Pr(S')} + \sum_i \log \frac{\Pr(w_i|S)}{\Pr(w_i|S')} \tag{1}$$

where the terms in the last sum are determined by the training sequence. This is called the log-likelihood ratio. In summary, we start by estimating the ratio of spam that is received. Current estimates say that approximately 80% of all incoming emails are spam, but it is common to assume no a priori knowledge of this ratio. Then, $\Pr(w|S)$ and $\Pr(w|S')$ are estimated by looking at many documents that we know are spam or that we know are not spam. Then we take the received document and compute the log-likelihood ratio (1). If the log-likelihood ratio is above a certain threshold, the email is regarded as spam. Choosing the threshold 0 means that we regard it as spam if the probability that it is spam is larger than the probability that it is not spam.

5.6 Hybrid Filters

Using several different methods is better than using only one when determining if an email is spam or legitimate. Depending on if one test fails or passes it will contribute to an overall score for the email. The scoring can be weighted depending on how effective one method is regarded to be. If the total score is higher than a certain threshold, an email will be regarded as spam. An open source implementation of a filter like this is SpamAssassin.

Exercises

Exercise 501 *Assume that you have found an open mail relay. Explain how you could use the SMTP protocol to send an email from any sender to any receiver.*

Exercise 502 *What part of an email is protected by DKIM?*

Exercise 503 *What part of an email is protected by SPF?*

Exercise 504 *Hashcash is implemented by the anti-spam software SpamAssassin. Different levels of credibility is given depending on the variant of hashcash used. Seven different credibility levels are specified as:*

```
Contains valid Hashcash token (20 bits)
Contains valid Hashcash token (21 bits)
                    ⋮
Contains valid Hashcash token (25 bits)
Contains valid Hashcash token (>25 bits)
```

What is the difference between them and why are they given different credibility?

Exercise 505 *Consider the following "received" header example:*

```
Received:from wikieditor.org (mailgw.riksdagen.se [194.52.83.65])
by mail1.ddg.lth.se; Fri, 16 Sep 2011 02:56:02 +0200
```

What do the different parts mean?

Exercise 506 *Both greylisting and nolisting assume that the MTA used by spam software is not implemented correctly or according to the standard. Which assumptions are made? Are they reasonable?*

References

[1] D. Crocker, T. Hansen, and M. Kucherawy. DomainKeys Identified Mail (DKIM) Signatures. RFC 6376 (Draft Standard), September 2011. Available at: *http://www.ietf.org/rfc/rfc6376.txt*.

[2] DMARC.org. Domain-based Message Authentication, Reporting and Conformance (DMARC). Available at: *http://www.dmarc.org/draft-dmarc-base-00-01.html*.

[3] R. Gellens and J. Klensin. Message Submission for Mail. RFC 4409 (Draft Standard), April 2006. Available at: *http://www.ietf.org/rfc/rfc4409.txt*.

[4] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. Obsoleted by RFC 5321, updated by RFC 5336, Available at: *http://www.ietf.org/rfc/rfc2821.txt*.

[5] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), October 2008. Available at: *http://www.ietf.org/rfc/rfc5321.txt*.

[6] J. Postel. Simple Mail Transfer Protocol. RFC 821 (Standard), August 1982. Obsoleted by RFC 2821, Available at: *http://www.ietf.org/rfc/rfc821.txt*.

[7] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408 (Experimental), April 2006. Available at: *http://www.ietf.org/rfc/rfc4408.txt*.