What all the stuff in email headers means—and how to sniff out spoofing

Parsing email headers needs care and knowledge—but it requires no special tools.

JIM SALTER - 8/7/2019, 7:30 AM - Are Technica

Come to think of it, maybe you shouldn't open this one at all.

I pretty frequently get requests for help from someone who has been impersonated—or whose child has been impersonated—via email. Even when you know how to "view headers" or "view source" in your email client, the spew of diagnostic wharrgarbl can be pretty overwhelming if you don't know what you're looking at. Today, we're going to step through a real-world set of (anonymized) email headers and describe the process of figuring out what's what.

Before we get started with the actual headers, though, we're going to take a quick detour through an overview of what the overall path of an email message looks like in the first place. (More experienced sysadmin types who already know what stuff like "MTA" and "SPF" stand for can skip a bit ahead to the fun part!)

From MUA to MTA, and back to MUA again

The basic components involved in sending and receiving email are the Mail User Agent and Mail Transfer Agent. In the briefest possible terms, an MUA is the program you use to read and send mail from your own personal computer (like Thunderbird, or Mail.app, or even a webmail interface like Gmail or Outlook), and MTAs are programs that accept messages from senders and route them along to their final recipients.

Traditionally, mail was sent to a mail server using the Simple Mail Transfer Protocol (SMTP) and downloaded from the server using the Post Office Protocol (abbreviated as POP3, since version 3 is the most commonly used version of the protocol). A traditional Mail User Agent—such as Mozilla Thunderbird—would need to know both protocols; it would send all of its user's messages to the user's mail server using SMTP, and it would download messages intended for the user from the user's mail server using POP3.

As time went by, things got a bit more complex. IMAP largely superseded POP3 since it allowed the user to leave the actual email on the server. This meant that you could read your mail from multiple machines (perhaps a desktop PC and a laptop) and have all of the same messages, organized the same way, everywhere you might check them.

Finally, as time went by, webmail became more and more popular. If you use a website as your MUA, you don't need to know any pesky

SMTP or IMAP server settings; you just need your email address and password, and you're ready to read.

Ultimately, any message from one human user to another follows the path of MUA ⟶ MTA(s) ⟶ MUA. The analysis of email headers involves tracing that flow and looking for any funny business.

TLS in-flight encryption

The original SMTP protocol had absolutely no thought toward security—any server was expected to accept any message, from any sender, and pass the message along to any other server it thought might know how to get to the recipient in the `To:` field. That was fine and dandy in email's earliest days of trusted machines and people, but it rapidly turned into a nightmare as the Internet scaled exponentially and became more commercially valuable.

It's still possible to send email with absolutely no thought toward encryption or authentication, but such messages will very likely get rejected along the way by anti-spam defenses. Modern email typically is encrypted in-flight, and signed and authenticated at-rest. In-flight encryption is accomplished by TLS, which helps keep the content of a message from being captured or altered in-flight from one server to another. That's great, so far as it goes, but in-flight TLS is only applied when mail is being relayed from one MTA to another MTA along the delivery path.

If an email travels from the sender through three MTAs before reaching its recipient, any server along the way can alter the content of the message—TLS encrypts the *transmission* from point to point but does nothing to verify the authenticity of the content itself or the path through which it's traveling.

SPF—the Sender Policy Framework

The owner of a domain can set a TXT record in its DNS that states what servers are allowed to send mail on behalf of that domain. For a very simple example, Ars Technica's SPF record says that email from `arstechnica.com` should only come from the servers specified in Google's SPF record. Any other source should be met with a `SoftFail` error; this effectively means "trust it less, but don't necessarily yeet it into the sun based on this alone."

SPF headers in an email can't be completely trusted after they're generated, because there is no encryption involved. SPF is really only useful to the servers themselves, in real time. If a server knows that it's at the outside boundary edge of a network, it also knows that any message it receives should be coming from a server specified in the sender's domain's SPF record. This makes SPF a great tool for getting rid of spam quickly.

DKIM—DomainKeys Identified Mail

Similarly to SPF, DKIM is set in TXT records in a sending domain's DNS. Unlike SPF, DKIM is an authentication technique that validates the content of the message itself.

The owner of the sending domain generates a public/private key pair and stores the public key in a TXT record on the domain's DNS. Mail servers on the outer boundary of the domain's infrastructure use the private DKIM key to generate a signature (properly an encrypted hash) of the entire message body, including all headers accumulated on the way out of the sender's infrastructure. Recipients can decrypt the DKIM signature using the public DKIM key retrieved from DNS, then make sure the hash matches the entire message body, including headers, as they received it.

If the decrypted DKIM signature is a matching hash for the entire body, the message is likely to be legitimate and unaltered—at least as

verified by a private key belonging only to the domain owner (the end user does not have or need this key). If the DKIM signature is invalid, you know that the message either did not originate from the purported sender's domain or has been altered (even if only by adding extra headers) by some other server in between. Or both!
This becomes extremely useful when trying to decide whether a set of headers is legitimate or spoofed—a matching DKIM signature means that the sender's infrastructure vouches for all headers below the signature line. (And that's *all* it means, too—DKIM is merely one tool in the mail server admin's toolbox.)

DMARC—Domain-based Message Authentication, Reporting, and Conformance

DMARC extends SPF and DKIM. It's not particularly exciting from the perspective of someone trying to trace a possibly fraudulent email; it boils down to a simple set of instructions for mail servers about how to handle SPF and DKIM records. DMARC can be used to request that a mail server pass, quarantine, or reject a message based on the outcome of SPF and DKIM verification, but it does not add any additional checks on its own.

Analyzing a sample email

Below you'll find a set of real headers from a real email. They show a fairly convoluted—but legitimate—path from an AOL account to a locally hosted Exchange server. They've been heavily redacted, with IP addresses, hostnames, and timestamps altered, but they're still intact enough for analysis.

We'll break it up into chunks, but we're reading those chunks strictly in order from top to bottom. Each server along the path prepends its own header to the *top* of the raw email body, above the headers of all servers that came before it. So as you read these, you're starting from

the final destination MTA and working your way down toward the MTA that first accepted the message from the sender's MUA.

The first page of headers from our sample email show us transport from a locally hosted Exchange server (top) through a private, third-party filtering service (the Postfix headers at the bottom). ***Jim Salter***

This second page of headers (still from the same sample email) is where things get interesting. Microsoft's Outlook.com has received the email from a Yahoo mailserver, sending (legitimately) on behalf of an AOL user. ***Jim Salter***

The third and final (bottom) page of headers tells us about the originating sender: an iPhone, using a Web service (or possibly an app front-end for a Web service). ***Jim Salter***

First group: Locally hosted Exchange infrastructure

```
Received: from REDACTED01.exch.domain.local
(10.4.3.88) by
REDACTED01.exch.domain.local (10.4.3.88) with
Microsoft SMTP Server
(version=TLS1_2,
cipher=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P521)
id
15.1.1713.5 via Mailbox Transport; Tue, 06 Aug 2019
09:51:59 -0400

Received: from REDACTED01.exch.domain.local
(10.4.3.88) by
```

```
REDACTED01.exch.domain.local (10.4.3.88) with
Microsoft SMTP Server
(version=TLS1_2,
cipher=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P521)
id
15.1.1713.5; Tue, 06 Aug 2019 09:51:59 -0400

Received: from redacted2.privatedomain.net
(10.11.74.70) by
REDACTED01.exch.domain.local (10.4.3.89) with
Microsoft SMTP Server
(version=TLS1_2,
cipher=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P521)
id
15.1.1713.5 via Frontend Transport; Tue, 06 Aug
2019 09:51:59 -0400
```

Looking at the first few header blocks on this message, we can see it was received by an on-premises Microsoft Exchange server run by the recipient's organization (ie, not on Office365 or Azure). We can also see that the message is being sent via encrypted SMTP along the way on these two hops and what ciphers were used.

We know this is a Microsoft Exchange server both by the (private) hostname, and more directly, by the "Microsoft SMTP Server" application name listed in each header block.

Note each IP address in these blocks is private. Any IP address that begins with `10.` is private and not publicly routable per IETF RFC1918. That indicates that this leg of the email's journey was over a privately managed network and not over the actual internet. We haven't yet left the local infrastructure.

When we make it to the third header block—received from `redacted2.privatedomain.net`—we mark the transition from the

local Exchange server to a mail filtering service. We can see that although we still haven't left the private network space, we've moved to what is likely a different subnet—from the local `10.11.x.x` to `10.4.x.x`—which may indicate either a VPN tunnel to a different location, or simply a different VLAN on the same overall local network. Because the IP address is private, there's no way for us to know much more than that. We can also see that this leg of the journey wasn't strictly SMTP; it was "via Frontend Transport," a Microsoft-Exchange specific protocol.

Second group: A third-party filtering service

```
Received: from redacted3.privatedomain.net
(redacted3.privatedomain.net [10.6.56.120])
 (using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-
SHA384 (256/256 bits))
 (No client certificate requested)
 by redacted4.privatedomain.net (Postfix) with
ESMTPS id 9935F120006
 for <redacted@recipient.tld>; Tue, 06 Aug 2019
06:52:59 -0700 (PDT)

Received: from redacted4.privatedomain.net
(redacted4.privatedomain.net
[10.11.72.70])
 (using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-
SHA384 (256/256 bits))
 (No client certificate requested)
 by redacted5.privatedomain.net (Postfix) with
ESMTPS id 6AB16120005
 for <redacted@recipient.tld>; Tue, 06 Aug 2019
06:52:59 -0700 (PDT)</redacted@recipient.tld></
redacted@recipient.tld>
```

Although we're still on a private network, as seen by the `10.x.x.x` IP addresses, we can see that we've left the Exchange ecosystem—the next MTA in the chain is running Postfix, a common Unix-based MTA (I'm using "Unix-based" here as a stand-in for Linux, BSD, and all other *nix-y operating systems; I am also very aware of the differences between a kernel and a distro so please let's not bog down into a semantic argument). We can also guess from the sudden change in server time zone that we're probably no longer on the local network. This hop's Postfix server is on Pacific time, while the local Exchange servers were on Eastern time. This is most likely a third-party mail filtering service, which may strip spam and/or malware attachments from incoming email.

Third group: outlook.com mail service

```
Received: from NAM03-CO1-
obe.outbound.protection.outlook.com
(mail-
co1nam03lp2053.outbound.protection.outlook.com
[104.47.40.53])
 (using TLSv1.2 with cipher ECDHE-RSA-AES256-SHA384
(256/256 bits))
 (No client certificate requested)
 by east.smtp.mx.exch.serverdata.net (Postfix) with
ESMTPS id 299186001F
 for <redacted@recipient.tld>; Tue, 06 Aug 2019
06:52:59 -0700 (PDT)

Received: from CO1NAM03FT022.eop-
NAM03.prod.protection.outlook.com
(10.152.80.52) by CO1NAM03HT180.eop-
NAM03.prod.protection.outlook.com
(10.152.80.254) with Microsoft SMTP Server
(version=TLS1_2,
```

```
cipher=TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384) id
15.20.1987.11; Tue, 06 Aug 2019 13:52:57 +0000</
redacted@recipient.tld>
```

Now, we've finally gotten to something public—the recipient in this case is using Microsoft's `outlook.com` hosting service to handle edge transport of their mail. This means we're about to get to something meaty: `outlook.com`'s assessment of the DKIM and SPF validity of the sending domain.

Remember, *any* change in the email body—including a change or addition to the headers so far—would invalidate the signature. A passing DKIM grade from `outlook.com` here will indicate that the message both (probably) originated from a sender within the domain in the `From:` header, and has (probably) not been altered since it left that domain's infrastructure. A DKIM fail would indicate that the message is either completely fake or was tampered with prior to `outlook.com` receiving it.

Fourth group: DKIM, SPF, and DMARC results

```
Authentication-Results: spf=pass (sender IP is
74.6.132.229)
smtp.mailfrom=aol.com; redacted recipient domain;
dkim=pass (signature was
verified) header.d=aol.com;redacted recipient
domain; dmarc=pass action=none
header.from=aol.com;compauth=pass reason=100
Received-SPF: Pass (protection.outlook.com: domain
of aol.com designates
74.6.132.229 as permitted sender)
receiver=protection.outlook.com;
client-ip=74.6.132.229;
helo=sonic306-30.consmr.mail.bf2.yahoo.com;
```

```
Received: from
sonic306-30.consmr.mail.bf2.yahoo.com
(74.6.132.229) by
CO1NAM03FT022.mail.protection.outlook.com
(10.152.80.182) with Microsoft SMTP
Server (version=TLS1_2,
cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) id
15.20.1987.11 via Frontend Transport; Tue, 06 Aug
2019 13:52:57 +0000
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/
relaxed; d=aol.com; s=a2048;
[potentially identifying information redacted]
```

This is `outlook.com`'s header detailing their SPF/DKIM assessment of the message that was sent. The message claimed to be from a user `redacted@aol.com`. The SPF record for `aol.com` includes `yahoo.com`'s SPF records, and the server that `outlook.com` received the message from—`74.6.132.229`—is a Yahoo mail server, so the message passes SPF validation.

More importantly for us, we see that DKIM verification also passed. This means that we can be pretty certain that the message in question really did come from an `@aol.com` mail account—it was signed by a legitimate `aol.com` mail server on its way out and was (probably) not tampered with in transit between the `aol.com` and `outlook.com` networks.

Fifth group: The MTA that first received the message, and the MUA that sent it

```
Received: from sonic.gate.mail.ne1.yahoo.com by
sonic306.consmr.mail.bf2.yahoo.com with HTTP; Tue,
06 Aug 2019 13:52:55 +0000

Date: Tue, 06 Aug 2019 13:52:55 +0000 (UTC)
```

```
From: redacted sender <redacted@aol.com>
To: redacted recipient <redacted@redacted.tld>
Message-ID: <redacted-id@mail.yahoo.com>
Subject: redacted
```

Although we're reading it last, this is the *first* set of headers tacked onto the original email, put there by the Yahoo webmail server that received it from its original source.

Yahoo's server didn't receive the message via an MUA using standard SMTP; instead, it created the message from input it received on a Web application, as noted by the `with HTTP` line in the header.

The `From`, `To`, and `Subject` lines are generally created by whoever wrote the originating message and aren't normally very trustworthy—there's nothing inherent in the SMTP protocol that keeps you from sticking `potus@whitehouse.gov` or similar in the `From` field. But in this case, we've already verified that the `From:` claims an `aol.com` email address and that `outlook.com` received the message from a mail server that `aol.com` says is allowed to send `@aol.com` email. Since we can also see that it was signed with a private key that only `aol.com` should possess, we can be pretty certain that it comes from where it said it does and that its content has almost certainly not been manipulated.

AOL does not allow you to monkey with a `From:` address when sending email; its interface only allows you to use a "screen name" under your control. This can be the primary screen name for your account or one of up to six others that can be added or deleted at will. Each screen name's email address is a fully working email address tied to the user's account and personal identifying information, however, so this would be good news for anyone who's prepared to subpoena an ISP in order to identify a harassing sender.

Finally, Yahoo's webmail server tells us a little of what it knows about the device that logged in to its Web application to create and send the message:

```
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="----=_Part_redacted"

References: <redacted-id.ref@mail.yahoo.com>
X-Mailer: WebService/1.1.13837 aoljas Mozilla/5.0
(iPhone;
CPU iPhone OS [redacted] like Mac OS X)
AppleWebKit/[redacted version number] (KHTML,
like Gecko) Version/[redacted] Mobile/[redacted]
Safari/[redacted]</redacted-id.ref@mail.yahoo.com>
```

The `X-Mailer` header here appends the `User-Agent` information from the device that contacted the Yahoo webmail server. In this case, the information points to an end user using an iPhone to send the message. In some cases, you can see the original IP address of the end user who sent the message; in this case, Yahoo/AOL decided to protect that information—the service itself knows what IP address the end user's device was using, but they're not likely to tell the recipient without a valid subpoena.

How much of this information can we trust?

One big problem with analysis of email headers is that they, too, can be faked by malicious servers along the way. You always know that the very topmost header on a raw email is absolutely something added by your own mail server, since that was the last one in the chain. But beyond that, any malicious mail server could have freely altered any header information from servers that came before it.

In this case, DKIM gives us considerably more confidence in the validity of the headers we really care about. Since the recipient here uses `outlook.com` as its edge transport for their email, and `outlook.com` does DKIM signature validation of the email it receives, we know that the entire content of the email, including the headers below `outlook.com`'s bottom header, *should* be valid.

With that said, we can only trust that DKIM pass line if *we're* the ones using `outlook.com` as edge transport, *and* we can verify that all the server headers above `outlook.com` belong to our own trustworthy infrastructure. If you're coming into this as a third party, you can and should take the actual DKIM signature (which was present, but we redacted from these sample headers) and verify it against the entire email, including headers, below that one, using the sender's public key (which can be looked up in the sending domain's public facing DNS).

In the absence of DKIM signatures, we would not be able to trust *any* of the headers below the ones prepended by our own mail servers—or even, necessarily, the content of the email itself—and would need to go through the long process of manually contacting the operators of each mail server along the way, asking if the header lines matched corresponding entries in their SMTP server logs.

Is this enough to identify the person who sent it?

You can very rarely get enough information from email headers alone to positively identify a sender. The goal typically is to narrow down the potential range of senders and identify the exact information you would need to demand from an ISP by way of subpoena. If we needed to positively identify the sender in this case, we would have the potential to request a court order for further information from two entities: Yahoo and AOL.

Given the message ID (redacted here), the timestamp of receipt, and the name of the mail server, Yahoo would be able to provide the IP address used by the device that sent the message. That IP address could then be tied to a particular ISP, and that ISP could be subpoenaed to identify the billing information of the user to whom that IP address was allocated at the time the message was created. If the IP address Yahoo provides is an unsecured Web proxy, that's an unhelpful dead-end; but if it's an end-user IP address, it'll provide a reasonably strong correlation to the customer whose account it was allocated to.

In this case, since the message came from an AOL user and we can be fairly certain that the sending address was a screen name associated directly with an account, a lawyer might instead (or additionally) try to subpoena AOL for the customer information associated with that screen name.

Neither of these pieces of information—the sending IP address or the primary screen name of the AOL customer who sent it—is, by itself, an uncontestable smoking gun. But they can be strong evidence that helps to build a case.

original article:
https://arstechnica.com/information-technology/2019/08/ars-forensic-files-how-to-parse-through-e-mail-headers-and-spot-obfuscation/