/Email/overcome common problems

Email: it just works, right? Wrong. Gavin Montague takes a look at some of the common mistakes we make when using it in our applications and sites

Knowledge needed: Knowledge needed: basic programming for the web

Requires: Xxxx INFO NEEDED HERE

Project time: X

Despite its ubiquity, email tends to be given short shrift by developers. We'll spend days optimising web servers, work late refactoring our code and pull each other's hair out over text editor preferences, but no one thinks twice about poor old email. This attitude is particularly baffling when you consider that almost every site relies on it for:

- Confirming user identity
- Receipt/digital goods delivery
- Communication with users who aren't visiting right now
- Announcements/marketing
- Incoming sales enquires

Those all seem like pretty important tasks, but when was the last time you checked how many customers were receiving the email your site is sending out? In this article, we'll look at some of the most common problems with email and how you may not even know you're suffering from them.

The many factors that can influence whether or not your email can get to its recipient are usually lumped together as 'deliverability'. I tend to think of deliverability as being a mixture of:

- **Authentication:** are you who you claim to be in the 'sender' header?
- **Connotation:** are your messages of interest to the recipient, or are they spam?
- **Reputation:** are you a trustworthy sender? Do you honour recipients' and email service providers' (ESPs) requests?

Before we get to these, let's look at some general advice for dealing with email.

Don't send your own email

Email infrastructure can be remarkably tricky, and it's often a better idea to outsource your delivery to specialists. Bulk/marketing email services such as Campaign Monitor (www.campaignmonitor.com), MailChimp (http://mailchimp.com) and others can provide you with a full suite of tools for a fraction of a penny per email sent.

Similarly, for transactional email, there are services like Mandrill (http://mandrill.com), Mailgun (www.mailgun.com) and SendGrid (http://sendgrid.com), who allow you to send emails via their APIs or by exposing an authenticated SMTP server to you.

These third party services will confirm that you're following best practice. They usually have excellent analytic tools, too. However, you may want to take control of your own email because of cost, scaling issues or just because it's good to know what's happening behind the curtain.

Authentication

Email is older than the internet, dating back to the early 1960s. It was a more innocent time, when spam was nothing more than a delicious meat product and everyone with an email address could comfortably fit inside one tube train. The protocols describing how email works make no accommodation for verifying the identity of a sender, the content of the message, or any of the other issues that plague email today.

Over the years, ESPs and others have developed techniques for sifting out the spammers and phishers from the real people. In order to get legitimate messages through most filtering systems, there are some easy steps to take that will ensure you are conforming to these unofficial standards.

Most of the systems for validating the identity of a sender are reliant on DNS, the system that translates domain names to machine-readable IP addresses and back again. A discussion of DNS could fill a whole book. It's beyond the scope of our discussion. However, if you've ever registered a domain name and pointed it at a site, you probably know enough to understand all the terms here.

Lazy choices There are a wealth of SAAS companies you can outsource your email services to, but where's the fun in that?

What's being said? The feedback loops provided by major ESPs will give you valuable feedback about how your recipients view your company.

Killer insights The SpamAssassin rules wiki can give you an understanding of the characteristics of your emails that can be improved

Set up rDNS

While DNS points a domain name to an IP address, the rDNS record associates an IP address with a domain. This distinction is subtle but important. Anyone who owns a domain can point it at any IP address, but only the person in charge of an IP address can state what domain it should be associated with. In this way, you can think of the rDNS as being definitively 'true', and – because IP addresses are more tightly regulated that domain names – as having more authority.

An ESP is much more likely to reject email coming from an IP address that doesn't have a correct-looking rDNS entry. Normally, you won't be able to set the rDNS entry for your server, but a support ticket to your hosting company will usually be enough to get it sorted at their end.

When did you last check how many customers were receiving email from your site?

Sender Policy Framework

The Sender Policy Framework (SPF) is an open standard that uses DNS to whitelist the servers that can send email on behalf of a domain. As the owner of a domain, you may authorise servers to send email as "@yourdomain.com", publishing a TXT record in your DNS. The SPF record is simply: a list of servers that you allow to send messages; any subordinate systems that you cede control to; and an action to use when failures occur. You can check for an SPF record using the **dig** command:

```
$ dig -t txt leftbrained.co.uk | grep spf
... v=spf1 mx include:spf.messagingengine.com 198.211.127.241 -all
```

The records start by stating the protocol: **spf1**. It then lists three sets of servers that can send email for leftbrained.co.uk: any server in my MX records, any server listed in my email provider's SPF record, and finally the IP address of a remote box I maintain. If any other server sends a message claiming to be from my domain, it's caught by the final clause **all**, which is prefixed by a minus sign. This indicates that the message should be rejected by the recipient's ESP.

DKIM

SPF is quite a blunt tool and relies on you having exclusive sending rights on an IP address. If I were to include the IP address of a shared-hosting box in my SPF record then other tenants could send email claiming to be from my domain. For a finer level of control we can use DomainKeys Identified Mail (DKIM) to digitally sign our emails, and ensure that individual messages and their contents cannot be spoofed.

Setting up DKIM is a bit more involved than SPF, and entails installing software on your email server. We'll cover the theory behind the protocol in this article. Look to your system administrator, hosting company or nearest Unix geek for advice on setting up DKIM.

DKIM is often compared to a wax seal. When sending a letter, the writer seals the flap of the envelope with a complicated wax seal. Anyone intercepting the letter can't open it without breaking the seal, thus alerting the recipient to tampering. The interceptor can't send letters pretending to be from the original author because he doesn't have the stamp with which to seal the envelope.

DKIM uses public/private key cryptography to sign emails with a seal based on the content of the message, plus a private key held only by the sender. The public key is published as a DNS record, so recipient programs can decrypt and check the seal. If a message is signed with DKIM, you'll see a header like this:

```
DKIM-Signature: v=1; a=rsa-sha1; d=id.apple.com; s=id2048; c=relaxed/simple;
q=dns/txt; i=@id.apple.com; t=1364036350;
h=From:Subject:Date:To:
bh=Yv53eSPd+nzU8yApPHC+LUUEvAY=;
b=l+wp+NsL4+iycVM [...] k73Y4lyzb7c7RRQYZC88OlFhKH
```

This signature on an email from Apple describes the protocol used, the DNS record where the public key can be found, the fields used to generate the signature, and finally the encrypted signature itself. My email provider will retrieve the public key from id2048._domainkey.id.apple.com and use this, along with the hash fields, to verify that the message was indeed sent by Apple.

Setting up authentication doesn't improve your ability to get delivered per se, but it will mark your emails as being legitimately from your organisation. This will make it much harder for spammers to pose as you and, by extension, make your emails more trustworthy to ESPs.

>>

Use the source

When learning HTML and CSS you've probably made extensive use of 'View Source' to see how sites work. Email isn't quite as easy to dissect, but it's not far off.

Messages are sent between machines as plaintext and encoded attachments. Emails essentially have two parts: the head and the body. Today we're concerned about the head. Whatever program you use to read your email. it's likely to have an option to show the raw/source/original message. You'll use this view a lot when improving your email.

The head of an email is a series of name/value pairs that function very much like an HTML **<head>**. The information is mostly for the benefit of the program rather than the end user, but today it's of interest to us as a diagnostics tool to see what's going on as our messages bounce around the internet.

from

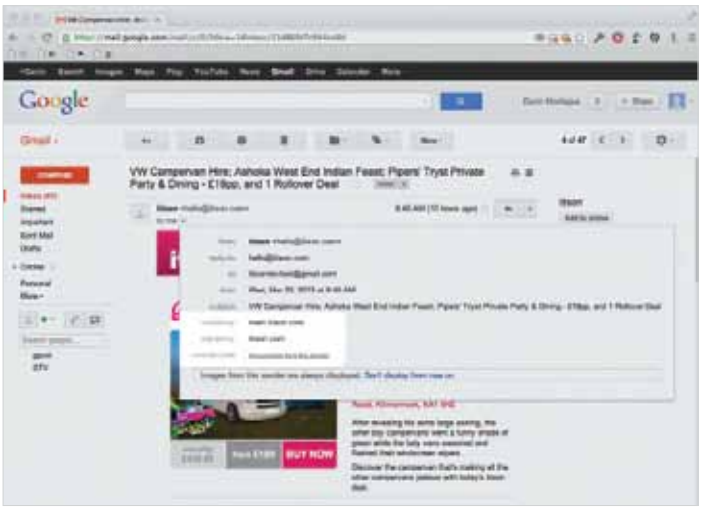
The name and address the message purports to be from. Note that email contains no intrinsic way of validating that a sender is who they claim to be: the **from** field is just a text-field reported by the sending server. Many phishing problems arise from this, and it's why we have to be concerned about authenticating our outbound messages.

reply-to

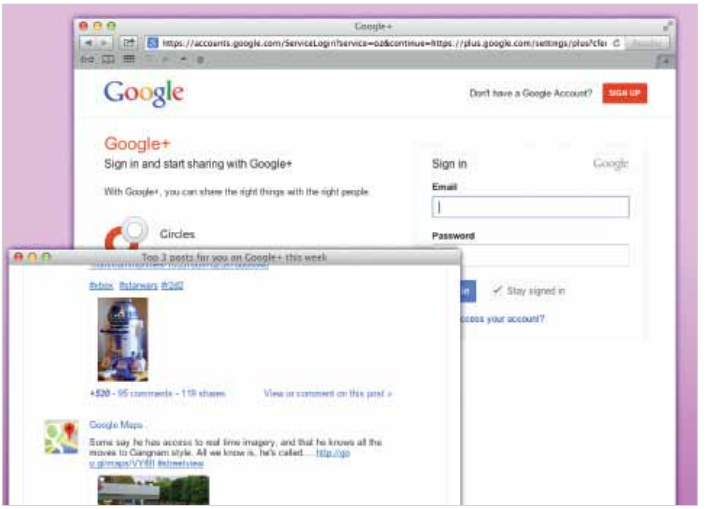
The address that the human reader should reply to. As mentioned in the main article, you should make sure your messages come with a valid reply-to address that is monitored by a human or intelligent set of mail-rules. Importantly, the reply-to is distinct from the return-path.

return-path

The **return-path** header looks similar to the **from** or **reply-to** field, but it indicates the address that should be used when there's a problem with delivery. Your return-path address should be distinct from your reply-to one as it will normally receive a much higher volume of automatic messages from recipient servers. You should regularly check your return-path mailbox for indications that you've been blacklisted or otherwise filtered by ESPs.



Audit trail In the above example, we can see Gmail indicating that our message is using SPF, DKIM and **list-unsubscribe**



Get-out clause Make it easy to leave. To unsubscribe from unwanted Google+ mail I'd need a password I don't know; net result – I just mark messages from Google as spam

Connotation

Identifying yourself is a good first step, but if your messages appear to contain spam then they'll be flagged as spam. Now you may say, "But we don't send spam!" Well, with the help of SpamAssassin (http://spamassassin.apache.org), let's have a look at what could make a message look spammy.

SpamAssassin is used to filter email before it reaches users' inboxes. At the simplest level, it reads messages and scores them against a battery of tests. A score above a certain threshold means spam. Some ESPs will include test results in the message header:

```
X-Spam-hits: BAYES_00 -1.9, RCVD_IN_DNSWL_HI -5, RP_MATCHES_RCVD
-0.704, LANGUAGES en, SA_VERSION 3.3.1
```

You can consult the SpamAssasin wiki (http://wiki.apache.org/spamassassin) to see what these tests mean. For example, **RCVD_IN_DNSWL_HI** indicates that the message comes from a whitelisted server and is very unlikely to be spam. We'll now look at some rules that you may fall foul of.

HTML has unbalanced body tags

Spammers tend to write bad HTML, so an email containing bad HTML is more likely to be spam. You should validate the markup of your emails as closely as that of your site. There are a slew of tests relating to the markup of your message rather than the content itself.

Images with 0-400 bytes of words

To get round language analysis, spammers sometimes send messages as a picture of text. Be wary of sending images in place of text content, or at least make sure to include a plaintext version too.

Subject line starts with buying

Perhaps your marketing team write an email letting customers know about a sale: "Buy our lawnmowers now and save 30 per cent!" Sadly, there are phrases so beloved by spammers that they should be avoided altogether. Use the SpamAssassin wiki (http://wiki.apache.org/spamassassin) as a style guide for phrases to steer clear of.

Contains a misspelling of word

Filters cope with the common spammer tactic of deliberately misspelling trigger terms like 'ca5ino' or 'perscriptions'. You should check your emails for spelling mistakes. Also watch out for things that look like trigger words. I had a problem where the random name assigned to one of my CDN servers at work looked too much like that of a male enhancement pill – our rejection rate skyrocketed!

Almost no test is individually strong enough to mark a truly legitimate message as spam. The effect is more like raindrops flowing together to create a flood. It's worth familiarising yourself with the sort of things that a filtering program will check, if only so you can prevent mistakes. Depending on your message volumes it may be worth setting up a local version of SpamAssasin or similar to test your messages against.

Reputation

While messages are judged individually on their content, ESPs also consider the history of the sending organisation. Over time, a server that repeatedly sends spam will find its way onto Real-time Blackhole Lists (RBL), such as Spamhaus (www.spamhaus.org). These publicly available lists are compiled and used by ESPs to pinpoint IP addresses that are not to be trusted.

Although it's possible to be removed from a blacklist by mending your ways, it's better to stay off them in the first place. The best way to stop yourself from being blacklisted by ESPs is to listen to what they're telling you.

The goal is to have our email read by users: as with the rest of the web, content is king

Feedback loops

Hotmail, Yahoo, AOL and other ESPs provide free tools that will help you monitor how your email is being received by their systems. By registering yourself as the 'owner' of an IP address, the ESPs will deliver near-real-time updates to you whenever one of their users junks your email.

Search for each provider's feedback loop, answer some questions to prove your ownership of a sending domain, and within a few hours you'll start to receive notices each time a user junks your messages.

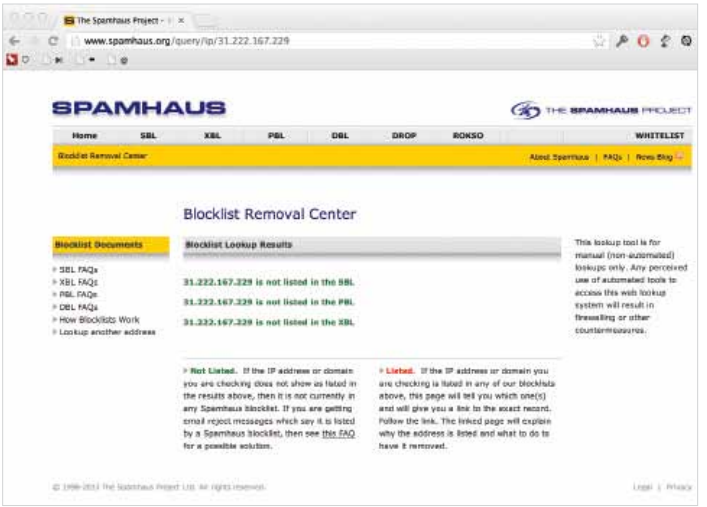
Gmail uses an alternative system called **list-unsubscribe**. The **list-unsubscribe** protocol allows senders to include an email address or HTTP endpoint as a header in their outbound email that should be pinged when the recipient wishes to stop receiving messages:

```
List-Unsubscribe: <mailto:list-unsubscribe@yourdomain.com>
```

To a recipient, feedback loops and **list-unsubscribe** are analogous. When they open a message, the webmail client will display a notice asking if they no longer wish to receive messages. Clicking this notice will send a message to your loop address or list-unsubscribe end-point.

These systems are important, because users simply don't trust 'unsubscribe' links. A large percentage will choose to end their relationship with you by marking your messages as spam.

They don't necessarily mean this in a bad way, but from their perspective unsubscribing and blacklisting have the same effect. They don't necessarily think of you badly. Many people will have found the latter option to be a more reliable and trusted method of stopping unwanted mail.



Bad meat Spamhaus is probably the most powerful RBL – use it to check the status of your IP addresses and take swift action if you find yourself blacklisted

The feedback loops and **list-unsubscribe** are a chance to redeem yourself. If you can respect the wishes of a junking user and stop sending messages, you'll not be penalised. When you receive an unsubscribe request from an ESP, or indeed anywhere, aim to act on it within 24 hours. If an ESP sees that you are repeatedly ignoring requests to stop mailing a particular address it'll come down on you even harder.

Consider how you could go about automating the unsubscribing process by writing a program that reads your feedback inbox and automatically unsubscribes complainants.

Make leaving easy

A related point is that one of the best ways of improving your deliverability is to make it very easy to leave. Imagine that I receive an email from a service I no longer want. Hopefully, the offending email will contain an unsubscribe link and I trust the sender enough to click on it. Imagine that the page now asks me to log in before unsubscribing: this is bad design.

As a user who wants to unsubscribe, I probably have very low engagement with the service, and it's unlikely that I can remember my user name and password. I could work out how to reset my password, jump through some more hoops and finally unsubscribe, but it's more likely I'll just go back to my inbox and hit 'Junk'. The more junk requests that service accrues, the worse its reputation will get.

Every email you send should explain to the user why they received it. Emails should include a prominent link that can unsubscribe users in no more than two clicks, and which requires no input on their part. Attach a unique, random token to each user in your database and pass this through in the unsubscribe link. This will give you enough information to individualise the request and ensure it is legitimate.

Don't send from no-reply

There's no excuse not to be friendly. Make sure your messages specify a reply-to address that is actually monitored and not just a black hole. A lot of users will try to get in touch with a sender by replying. They assume their message will be picked up. No amount of "don't reply to this address" can fix this.

Instead, specify a reply address that's monitored by either a person or a mail rule that will respond to incoming messages with a list of contact options for your company. This is trivial to implement and leads to far fewer frustrated support tickets from users: "I've emailed you about this five times already! Why didn't you answer?"

In conclusion

The ultimate goal is to have our email read by users. As with the rest of the web, content is king. If you want users to value your communications, you should only send them messages that they asked for and that will benefit them.

In the same way that a slow, awkward or unintuitive website will cost you traffic, so too will a badly-managed email system. I hope that I've given you some ideas about how you can improve yours.

Email analytics

If you want to improve any aspect of your email, the first thing you'll need is data. If you don't know how many people are opening your newsletters each month, how can you decide if changing the subject format has an effect?

Unfortunately, there's not an email equivalent to Google Analytics, the free site-metrics service most developers rely on to provide statistics on usage and visitors. Although some services do exist, such as Litmus (http://litmus.com), they tend to be quite costly. Luckily, it's quite easy to write a simple analytics program, mostly because what one can measure in email is limited to two techniques:

Image shims

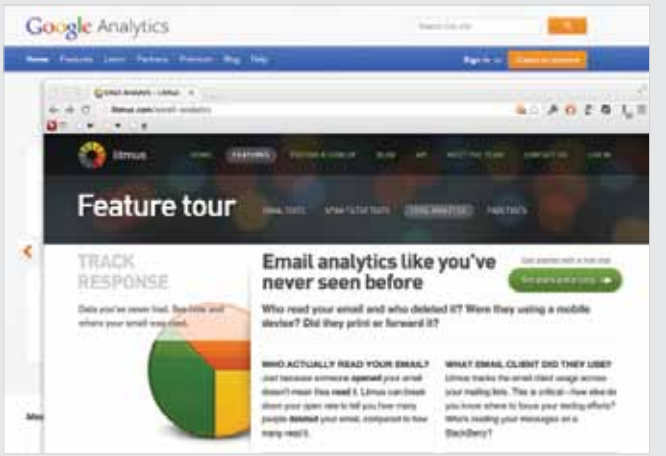
Also called a 'bug', this is an image tag that points to a server-side script in your program. When the user opens their email, the shim is loaded and gives your application a chance to capture data about the user and their email client. However, because most email clients will allow users to block images in their messages, it will always underestimate your true open rate.

Link tracking

The desired action for most emails is to bring the user back to your site. The best way to do this is via Google Analytics campaigns (http://netm.ag/campaign-242). This takes little effort and provides useful data. But because Google's integration with your app is limited, it's much harder to gather specific information about your users. For this, you'll need to collect your own information by injecting a 'redirect' path into your links. For example, my app may send an email about a sale to Bob and include this link:

```
http://sales.com/r.php?u=bob&source=banner&r=/lawnmowers
```

When he clicks, Bob is taken to the analytics script, which registers the hit and source before redirecting him to **/lawnmowers/sale**. From the end-user's perspective, nothing odd happened, but we can now start aggregating useful data on how people responded to our campaigns.



Acid test The likes of Litmus (http://litmus.com) eclipse Google Analytics for email metrics, but are pricey – so you're as well building your own simple program

About the author

Name Gavin Montague
Web http://leftbrained.co.uk
Twitter @gavinmontague
Areas of expertise Ruby, usability
Clients itison
What's the most trouble you've ever been in?
Xxxxx