INKY™

Which deployment model is right for me?

API vs Inline

Several startups now offer API-based solutions to enhance email security, particularly addressing the pervasive problem of phishing.

At INKY we've studied (and implemented) both API and inline deployment styles to assess their relative merits. This document provides our own analysis of the trade offs, and gives customers insight into why we no longer believe API-based approaches are sensible.

APIs Explained

When vendors tout an "API-based solution", they mean they access customer mail using a REST API like the Microsoft Graph API or Gmail API. The platform providers offer these APIs to make life easier for third party vendors: virtually all developers these days know how to use REST APIs, and interfaces like these work well with modern JavaScript frameworks.

This means that startups looking to add value to the email ecosystem don't need to hire people with specialized email protocol knowledge; the same developers who write, say, user interface code, can also develop the code to access email. This opens up the world of mail processing to a much larger potential pool of vendors — a good thing overall.

Advantages of API Access

Beyond the obvious advantages to the vendors, API approaches have some potential value to customers as well.

The main advantage is simplicity: an API solution is easy to understand and easy to deploy: conceptually, a customer admin user just needs to perform an oAuth authentication and the vendor's cloud services can begin processing the customer's email.

Additionally, customers sometimes get comfort from the fact that the vendor's service is running alongside the customer's mail flow, not in the middle of it. The feeling is that as long as the vendor's processing isn't intermediating the mail flow, the customer can't be impacted if the vendor's service is interrupted or somehow compromised.

API-based mail protection vendors trumpet these benefits, claiming "risk-free deployment in minutes" in their marketing materials.

Downsides of API Access

Unfortunately, as in all things, there is no free lunch. The simplicity of REST API access comes with some serious baggage.

The biggest challenge with API access is: since it's by definition not inline, there is no way to prevent mail from reaching the end user's inbox. For mail protection scenarios in particular, this a problem: what is the point of identifying a malicious email if that mail is going to get delivered anyway?

There are workarounds to this problem: one is to configure Exchange or G-Suite to deliver all mail to a temporary folder, then have the API-based third party software move mail from this folder to the inbox only after the mail has been analyzed and found to be safe.

This is what we in the technical community would charitably label "a hack." And before declaring victory here, it's important to realize two serious issues with this hack:

1. There's nothing to prevent an end user from interacting with a malicious mail; they can just go into the temporary folder to read it.

2. If the temporary folder is hidden to prevent problem #1, then the API-based third party service is once again critical to the delivery of mail to the inbox — thereby negating one of the primary advantages of using an API and not being inline in the first place.

API access is problematic in another regard as well: neither the Gmail nor the Microsoft APIs allow any modifications to be made to emails. API users can move mails between folders, but can't modify the contents of emails.

This is bad because it means there's no way for mail protection software to neutralize a malicious email. An API-based solution can't quarantine email, because it's not inline. So what should it do when it identifies a malicious email?

Inline mail protection systems like INKY can quarantine mail, of course, but can also protect users via less draconian measures as well. INKY (and other inline mail protection systems) can rewrite URLs in suspicious emails to route them through a proxy page or block them entirely. This obviously depends on the ability to modify the email, so it's infeasible for an API-based solution.

Uniquely, INKY can also add a color-coded informational banner to the top of the email: a red "Danger" banner for malicious content or a yellow "Caution" banner for unusual, sensitive, or suspicious content. This is very helpful for dealing with borderline cases, where there simply isn't enough information available to convict an email as malicious, but where we really want the user's guard to be up.

Here again, you need a deployment mechanism that allows modification of the email for this to work. It's worth pointing out that even the platform owners themselves — Google and Microsoft — only add such notifications as user interface elements in their own clients, which does nothing to help users using other clients like mail.app on iOS. Modifying the email itself is key here.

Finally, we've found these APIs to be brittle and not scalable — they are meant for add-on capabilities, not core mail functions. As a consequence, the platform owners change them frequently, often with breaking changes, and put very little effort into ensuring they are suitable for processing, say 100% of email for a 10,000 person company. The APIs are simply not meant for industrial-strength deployments across entire organizations. This is why Microsoft explicitly throttles both API calls and mailbox notifications.

In contrast, inline mechanisms rely on standards and software that have been refined for literally decades. When we deploy INKY inline, we're using standard SMTP protocols and software stacks that have been proven to scale to arbitrary sizes with real-world systems like Exchange, Postfix, and Sendmail. These standards address essential requirements around confidentiality (TLS) and reliability (queuing and a store-and-forward architecture).

Our analysis is that while API-based solutions initially seem safer, they are in reality much less likely to scale and perform reliably — so even from the standpoint of risk minimization, they are inferior.

The one remaining advantage of an API-based access method — that it doesn't require familiarity with mail protocols or specialized practitioners — is of no value to us. That's because our whole mission in life is to be experts on mail and mail protection. We hire developers who know everything there is to know about email and email standards, so why would we have them use an API meant as training wheels for developers who don't know these things?

Inline Deployment Variations

Given the decision to us inline deployments at INKY, the question remains: where exactly should INKY's processing be inserted?

Traditional mail protection vendors deploy as the so-called "MX" — the public-facing mail server that all incoming email from the Internet hits first. This advantages the mail protection vendor, because it gives them control: since they are the first to see and modify the email, all downstream systems must accommodate them — and they need not worry themselves with handling any upstream processes. It also means that doing anything with your mail requires you to go through them — which gives them an opportunity to "engage" with you (i.e., work on selling you up to more features).

For customers, however, MX deployments are painful: they require changing the public DNS records for all email, to route the mail to a third party. Furthermore, if that third party is down, the customer's mail simply won't be delivered!

This is why we generally deploy INKY inline but downstream from the customer's existing MX. The platform provider's MX can still perform basic spam filtering before handing the mail off to INKY. INKY can then perform its analysis and pass the mail back into the platform provider's infrastructure.

This is the best of both worlds for customers: it allows INKY to perform all the analysis and

modification it needs to, but without creating a dependency on INKY as a service provider. Why is this so? Because in this kind of deployment the MX server itself communicates with INKY using a timeout: it tries to send the mail to INKY via SMTP and if INKY doesn't accept the mail within, say, 10 seconds, it just carries on and delivers the mail normally, without INKY's analysis. This removes INKY's infrastructure from the critical path of mail delivery.

This deployment model also works for customers who already use a third party as the MX. For example, Proofpoint customers who are still getting phished — *we hear from these constantly* — can deploy INKY downstream from the Proofpoint MX and the extra zero-day phishing protection INKY provides.

Conclusion

We hope this discussion has shed some light on what is otherwise a rather confusing topic.

At INKY we started without a dog in this fight: we explored all three deployment methods (API, MX, and inline downstream) and landed on our standard model simply because it works the best for most customers.

Have questions about INKY or interested in seeing a live demo? Drop us a line.