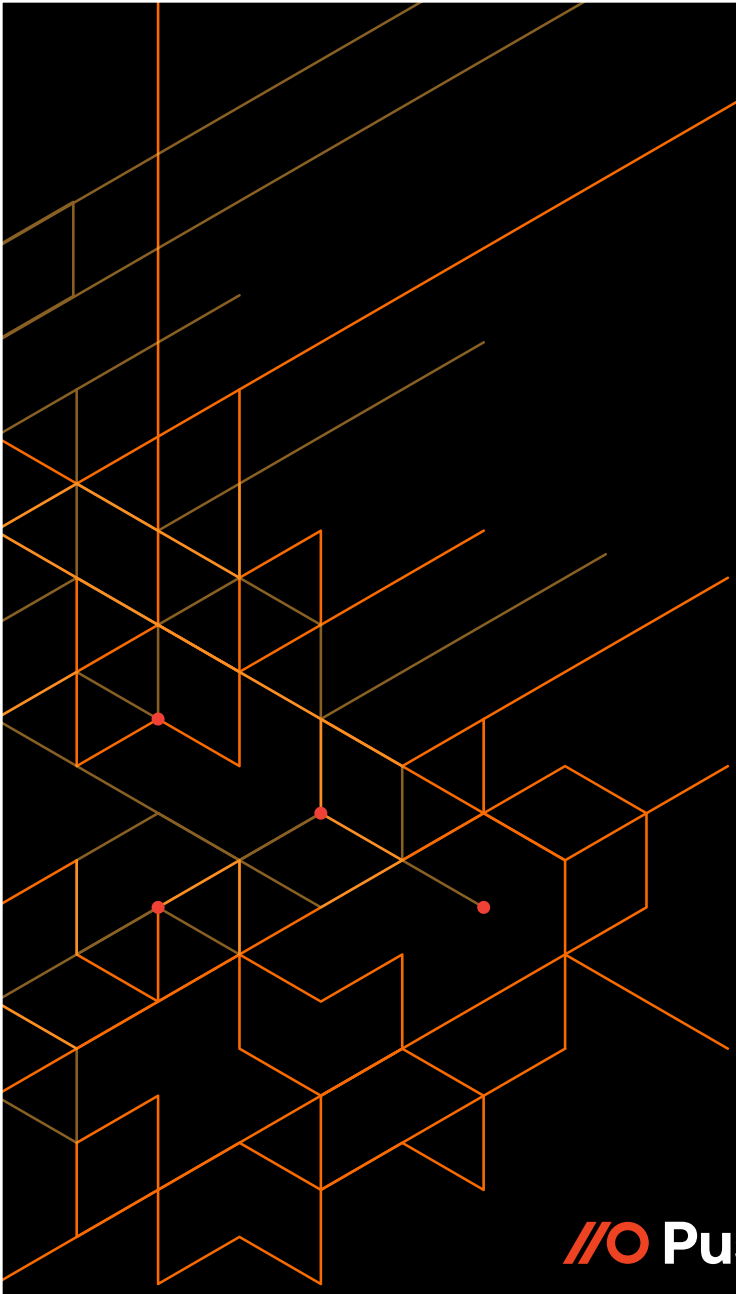
Push Security

The evolution of phishing attacks

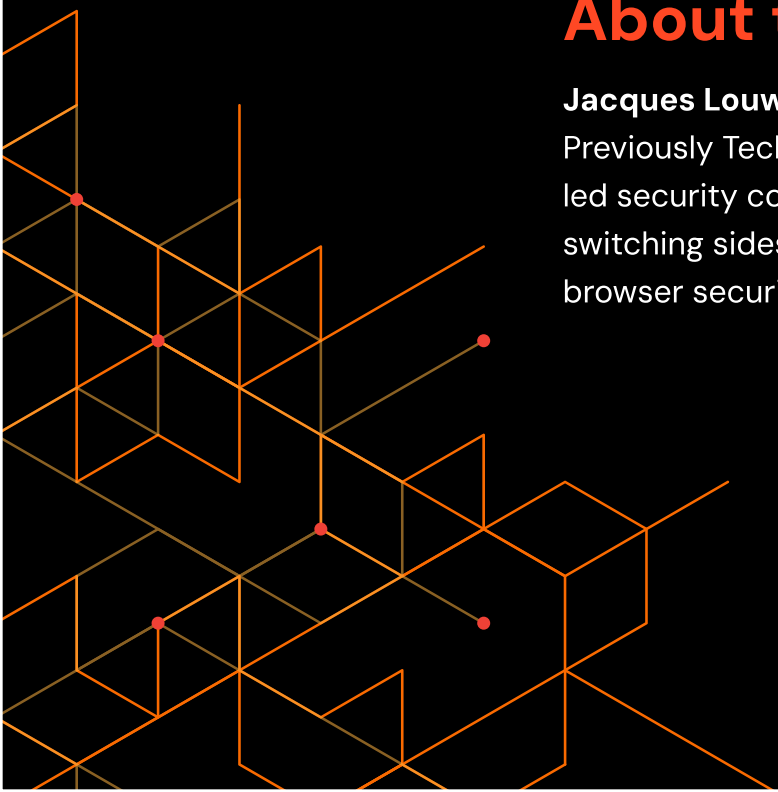
How modern phishing tools and techniques have changed the game — and what security teams can do to level the playing field.



Push Security

About the author

Jacques Louw is Co-founder and CPO at Push Security. Previously Technical Director at MWR InfoSecurity, a research-led security consultancy, he started as a red-teamer before switching sides to detection and response. Now, he builds browser security tools for blue teams at Push.



Push Security

Contents

Introduction

It's easy to write off phishing as unsophisticated and simplistic, particularly when we think back to the first generation of phishing attacks — static HTML pages purely designed to steal your username and password, linked directly from an email.

But modern phishing has changed a lot in the past decade or so. **MFA-bypassing Attacker-in-the-Middle (AitM) kits are table stakes** — anyone can pick up a copy of Evilginx and immediately blow past most email and network security solutions on the market.

But the most sophisticated attacks — the ones that usually hit the headlines in the form of major breaches — are doing much more than this. **The latest generation of fully customized AitM phishing kits are dynamically obfuscating the code that loads the web page**, implementing **bot protection through custom CAPTCHA**, and using **runtime anti-analysis features**, making them increasingly difficult to detect by the tools most enterprises are using to combat the problem.

The techniques used by attackers to deliver phishing lures are also more sophisticated. Groups like Scattered Spider have been seen using **malvertising** techniques, delivering phishing links via paid Google ads, while phishing campaigns are frequently encountered in **IM apps** (such as Slack and Teams), as well as **public messaging services** like LinkedIn and Reddit — bypassing email altogether.

The latest trends indicate that attackers are responding to increasingly hardened IdP/SSO configuration by circumventing MFA and passkeys, either by **downgrading to a backup (less secure) authentication method**, or sidestepping the legitimate auth process entirely through methods like **consent phishing**.

Attackers have also realized how much valuable data exists in Shadow SaaS, highlighted by major SaaS breaches impacting apps like **Snowflake**. This is driving **broader targeting of apps like Slack, Mailchimp, Postman, GitHub, and other commonly-used business apps directly** — bypassing Identity Provider (IdP) accounts like Microsoft, Google, and Okta that typically have more robust authentication controls and secure configuration options.

In this whitepaper, we'll take you through the evolution of phishing attacks, breaking down the key changes in attacker tooling and TTPs that got us to where we are today, explaining why they're so successful — and what security teams can do about it.

You can also check out our latest resource, the phishing detection evasion techniques matrix, where we've broken down the methods that modern phishing attacks are using, to help security teams to evaluate their security tools and capabilities and identify gaps. Like our SaaS attacks matrix, this will grow over time as new techniques are identified.

We'll signpost many of the phishing detection evasion techniques throughout this whitepaper.

Phase 1: Targeting
Identifying apps and users to evade security controls and achieve the shortest time-to-impact of a phishing attack.

Phase 2: Link delivery
Deliver links using phishing vectors that evade traditional security controls.

Phase 3: Link camouflage
Masking malicious links to prevent detection at the email, network proxy, or safe browsing layer.

Phase 4: TI evasion
Preventing TI feeds from flagging and blocking known-bad domains by masking or changing elements likely to be flagged.

Phase 5: Anti-analysis
Techniques to defeat automated "sandbox" analysis tools by preventing security teams and bots from accessing the page.

Phase 6: Page obfuscation
Obfuscating page elements to break detection signatures analysing page content and code.

Phase 7: Defeat MFA & CA
Defeat authentication and access controls in order to successfully execute the phishing attack.

Phase 8: Account takeover
Achieve a form of account takeover and conclude the identity attack, enabling further exploitation to take place.

Check out our phishing detection evasion techniques matrix on GitHub.

Learn more about Push's browser-based detection and response platform at **pushsecurity.com**

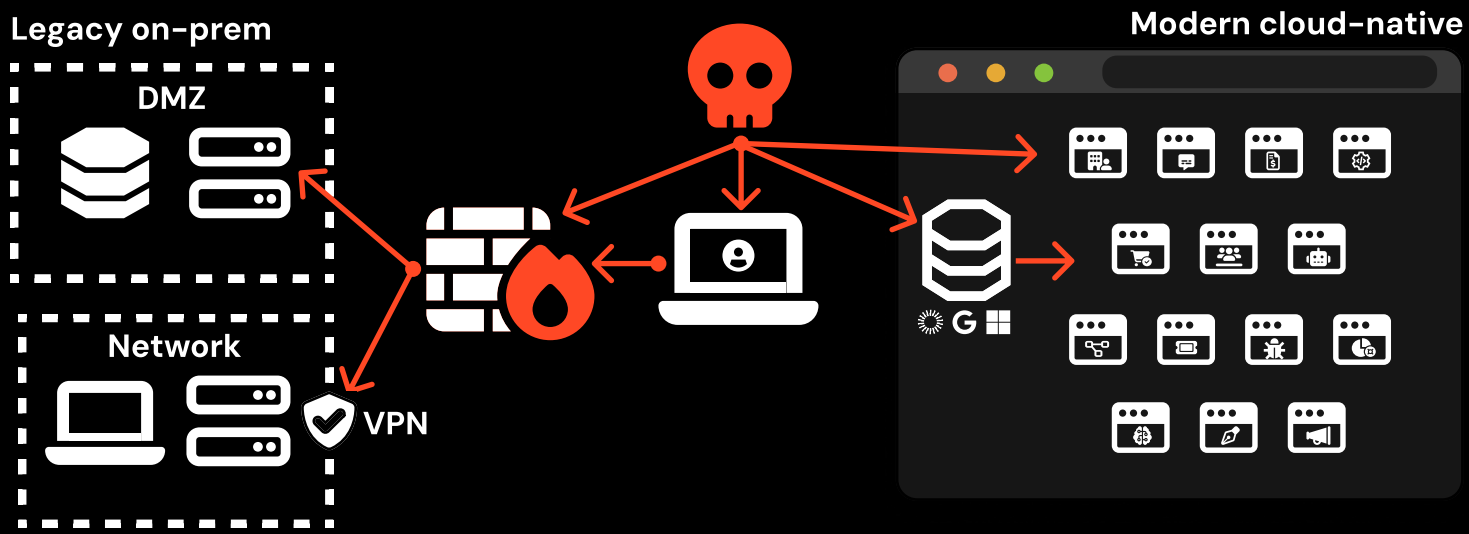
Gen 1: Classic phishing (in an endpoint-centric world)

A decade or so ago, phishing attacks took one of two forms:

- **Tricking a victim into installing malware** on their machine directly.
- **Tricking a victim into entering their credentials** (simply, a username and password, no MFA) into your static HTML webpage, which you could leverage to get to the point of installing malware.

Both were a valid and reliable way to get access to an internal network (in other words, breaching the network perimeter via an endpoint). You sent a user an email with a link to a static HTML webpage (most commonly a generic Exchange Web Access clone) that tricked them into giving you Active Directory credentials, which were used to login to an exposed remote desktop service or the victim's mailbox, which you could use as a foothold to install malware. In the early 2010s, this was a staple of "red teaming 101" for pentesters and hands-on-keyboard threat-actors alike.

A lot has changed since then. We're no longer running our businesses from the same kind of network. Modern business IT is much less on-premises castle-and-moat, and is instead a decentralized sprawl of SaaS and cloud services that users simply log into over the internet. And with the EDR revolution making malware-driven phishing attacks significantly harder than they used to be, identity is a much more viable option for attackers — which is reflected in the biggest breaches of the last few years (looking at you, Snowflake!).



Attackers are now targeting business applications directly over the internet via identities — without even touching the endpoint.

As identity became increasingly important, anti-phishing controls started to mature, and MFA in its more modern authenticator-based form became widespread (not just the SMS codes for your banking app that led early Scattered Spider to SIM swapping).

But before we dive into what phishing looks like today, let's recap its origins and how the email and network anti-phishing controls as we know them came to be.

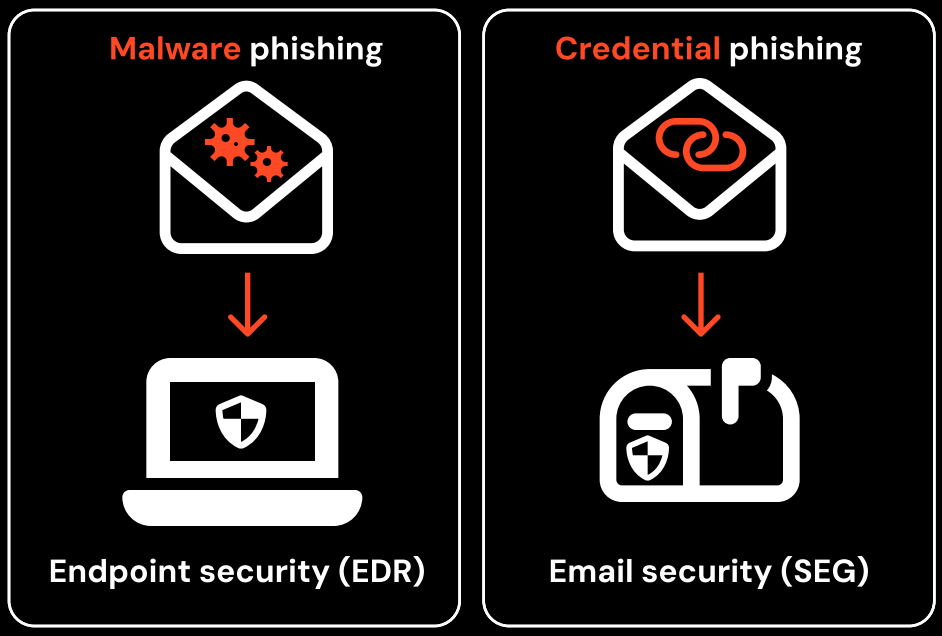
Setting the scene: A short history of phishing security

Phishing detection was (and still is) primarily email based. In the early 2010s, this was limited to inspecting attachments with embedded malware, and for about a decade adding macros to Word and Excel documents was the technique you were most worried about.

The original way that the industry (unsuccessfully) tackled malware phishing was through sandbox analysis. The TL;DR is that if you suspect an executable file to be malicious, you open it in a virtual machine and observe what it does using tracing tooling.

This quickly went out of fashion as sandbox evasion techniques (like setting a delay before the execution of malicious code) became standard practice in endpoint malware — it was clear that observing the live behavior in-situ using an agent was the only viable path forward. This is the insight that gave us CrowdStrike and the modern EDR industry.

When the EDR revolution hit, malware detection moved away from sandboxes to real-time detection on the device, leaving email-based controls to deal with credential attacks.



Email security: Blocklists, URL rewriting and "web sandboxes"

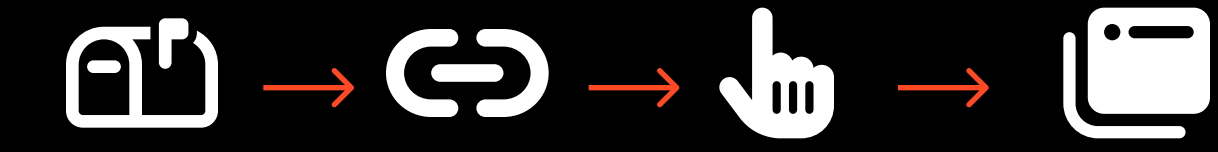
Early detection of credential phishing focused on employee training and adding "report phishing" buttons to Outlook. When a domain was flagged as malicious by a user, it was added a to a list of blocked domains to prevent other users from accessing the same known-bad page.

This was somewhat effective as at the time because there was no AWS, Route53, or Cloudflare, Let's Encrypt was the big new thing. Simple things like changing your phishing website's domain name and getting new SSL certificates required manual human input, and so genuinely slowed attackers down **(in a way that these things don't do today).**

Fast forward a few years, a new breed of phishing security solution is born analysing signatures for the content of phishing pages, not just looking for known-bad domains.

The caveat here is that you cannot simply load a link in an email to fetch the phishing page — because loading that link can cause unpredictable actions. Imagine for example your email security solution opened every link in all your emails — it would constantly be "unsubscribing", "confirming", and "opting-into" a whole load of things.

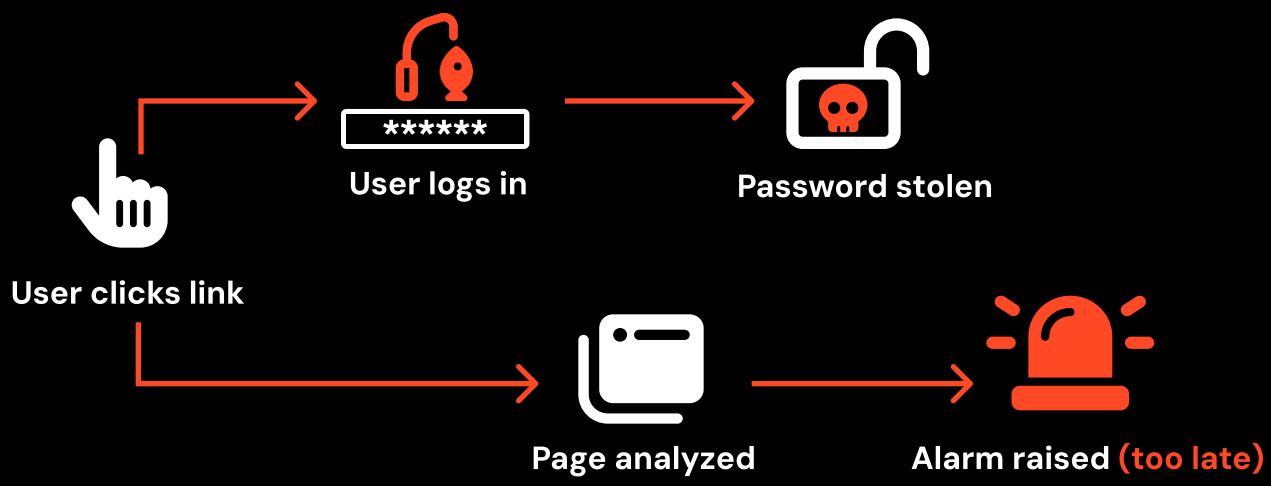
To get around this inability to open links until the user decides to click them, email security vendors landed on the technique of **URL rewriting**, which replaces the URL of a link in an email with a link they can monitor and which, when clicked, will forward you to the original link. Think of this almost like a link shortening service: **When the user clicks a link, the vendor clicks the same link and analyses the content in their sandbox.**



Since modern web pages are dynamic and require JavaScript to fully render the page, email solutions opted to open them in a headless browser (essentially a "**web sandbox**") and analyze the resulting DOM (basically the fully loaded web page). This way, you can take screenshots of the rendered page to do visual similarity checks, look for known bad scripts, or any other signature in the page source.

Once a site was identified as malicious, it could be blocklisted, preventing any users from accessing it. Security service providers were able to aggregate this threat intelligence, providing TI feeds or managed services to block pages as they were discovered.

This sounds useful, but the crippling shortcoming of this approach is that **none of this can be done ahead of time.** At best it can be done simultaneously, but mostly it happens after the fact. Users won't accept being asked to wait every time they click a link so the web page can be analysed in the sandbox before they are let through.



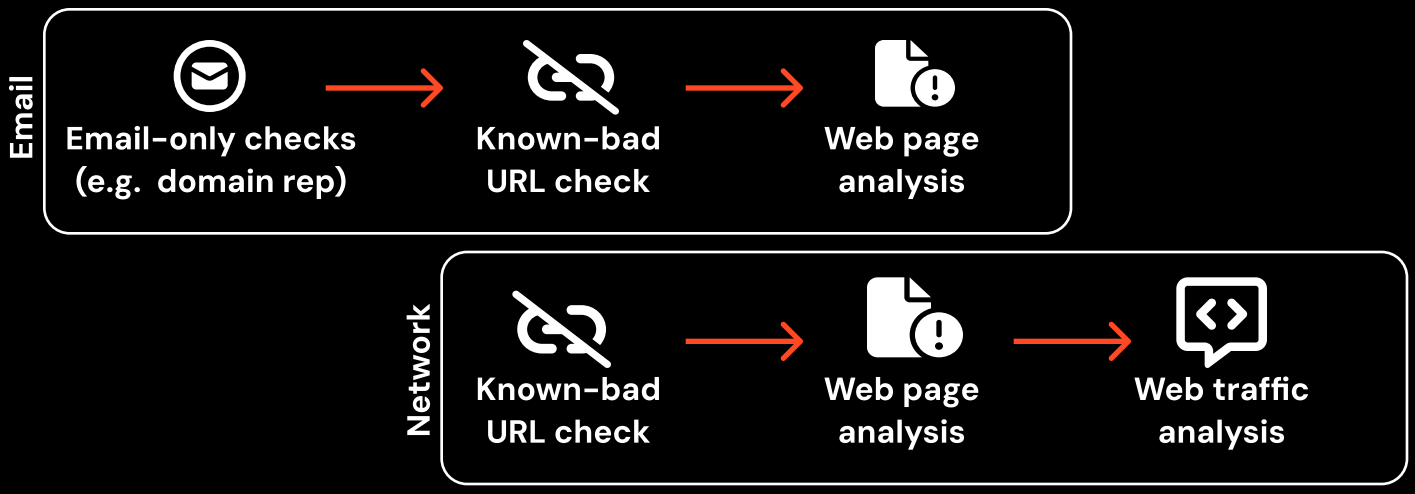
Email tools were always one step behind due to the lack of true real-time analysis.

Consequently, the first user that clicks a phishing link seen for the first time will be let through and hit the phishing site — and potentially fall victim.

The saving grace here is that if the sandbox analysis succeeds in detecting the link as phishing, the email solution can find other emails with the same or similar links and quarantine them, to prevent more users accessing the same link. And in the old days, resetting a phished AD account credential was pretty straightforward (though much less so in today's world of cloud IdPs and SaaS sprawl).

Network security: Web proxies and proactive scanning

Alongside email security controls, network security controls were introduced to inspect the page as it was loaded, as well as the network traffic resulting from the user's interactions with the page.



Network controls overlap with email solutions but with additional visibility of web traffic.

Network-based controls came in t**wo main forms: proactive scanning of pages** before they were sent to victims, and **using web proxies to intercept and analyze traffic** resulting from users interacting with a phishing page (aka. "break and inspect").

Proactive scanning of web pages

To bolster point-in-time scanning of pages as they are sent to would-be victims, security vendors started to monitor public data feeds to discover new websites going live on the internet.

The main data being consumed here are certificate transparency logs — each time a new TLS certificate is issued for a public domain, the CA issues an event on public transparency logs. If you're Google or Cloudflare and you're indexing the web and/or running a major DNS resolution service, you've got other great sources of information.

However you find out about a new website going live, you can then fetch that website, or load it in a web sandbox (i.e. headless browser) and analyse the site DOM. You can then do the same kinds of detections as the proxy or email solutions use, but with one massive advantage — you can detect phishing campaigns before links are even mailed to targets (meaning no sacrificial first victim). This can be very effective and today these feeds are built into mainstream browsers — most notably Google Chrome's integrated safe browsing service.

Traffic analysis via web proxy

If you're able to force all your workforce's web traffic through a proxy solution, it's possible for those solutions to inspect the content of web requests, and reconstruct the web page from this traffic. Since Gen 1 phishing attacks were simple static HTML pages, building detections for them was largely effective if you could afford the cost of the compute-time to "break and inspect" the traffic to drive detections. Modern solutions can do this relatively well, even without terrible latency implications.

That said, modern solutions also need to deal with distributed workforces, and typically that means a cloud-hosted, distributed proxy solution. A nasty side effect of this is that the proxy solution must do complete TLS-decryption of all web traffic to do useful detections. The result is that the more complex the page and the resulting traffic **(which is exactly what is about to happen in our timeline of phishing evolution)**, the less likely that you'll be able to analyze everything in real-time. This means most of this data is only really analyzed in post-hoc investigation scenarios.

The increasing complexity of web pages makes "break and inspect" less viable as a real-time source of phishing detections.

What changed?

Although there are some clear limitations and challenges, for the most part "classic" phishing scenarios were less of a problem for security teams than managing vulnerabilities and the cat-and-mouse game of EDR evasion and threat hunting. But then networks fundamentally changed with cloud transformation and SaaS adoption, increasing the size of the prize when it came to identity attacks and account takeover.

To take advantage of this emerging attack surface, attackers needed to ramp up their phishing efforts, and also had to contend with new controls like MFA. So, they adapted.

Gen 2: Phishing evolves to defeat detections and bypass MFA

Attackers looking to take advantage of the new world of cloud identity providers (IdPs) and SaaS services had to contend with two main challenges:

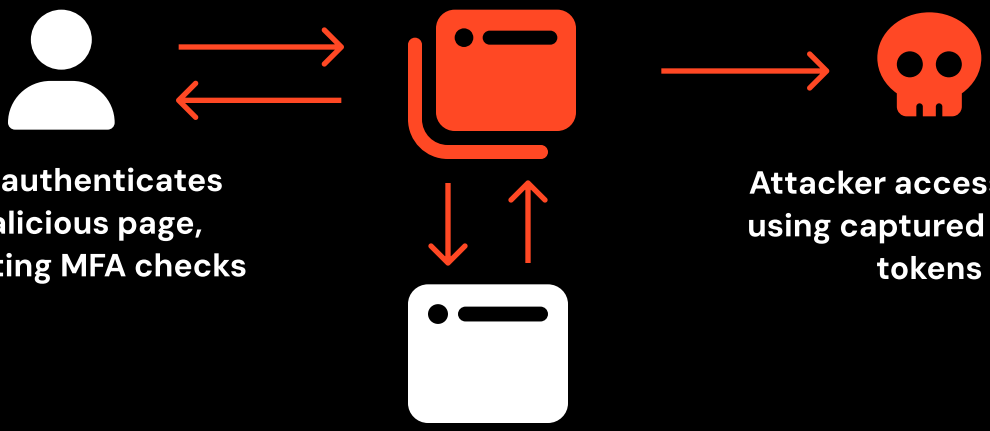
- **The broad adoption of MFA** (in particular the use of authenticator apps) especially on the most commonly targeted IdP platforms like Microsoft, Google and Okta.
- **The old "static website on a new domain" trick just wasn't working anymore** with automated and proactive website analysis.

Bypassing MFA with Attacker-in-the-Middle phishing

Because MFA is typically time-bound, attackers generally can't clone or copy the second factor (outside unscalable targeted attacks like SIM swapping). But, if you can get the victim to authenticate and create a real-session on your behalf, you don't need to intercept the MFA factor — just the resulting session token.

That's why MFA-bypassing Attacker-in-the-Middle phishing kits are the standard choice for attackers today. These work by intercepting the authenticated session created when a victim enters their password and completes an MFA check. To do this, the phishing website simply passes messages between the user and the real website — hence "Attacker-in-the-Middle".

In this scenario, the user is actually interacting with the real Microsoft website via the phishing "website". Part of the genius is that, in a way, this is actually less effort to do than creating a real phishing website — you can skip that and just pass messages. Once the user logs in, the attacker can take the session cookies and use that to access the victim's account.



Bypassing passkeys with downgrade attacks

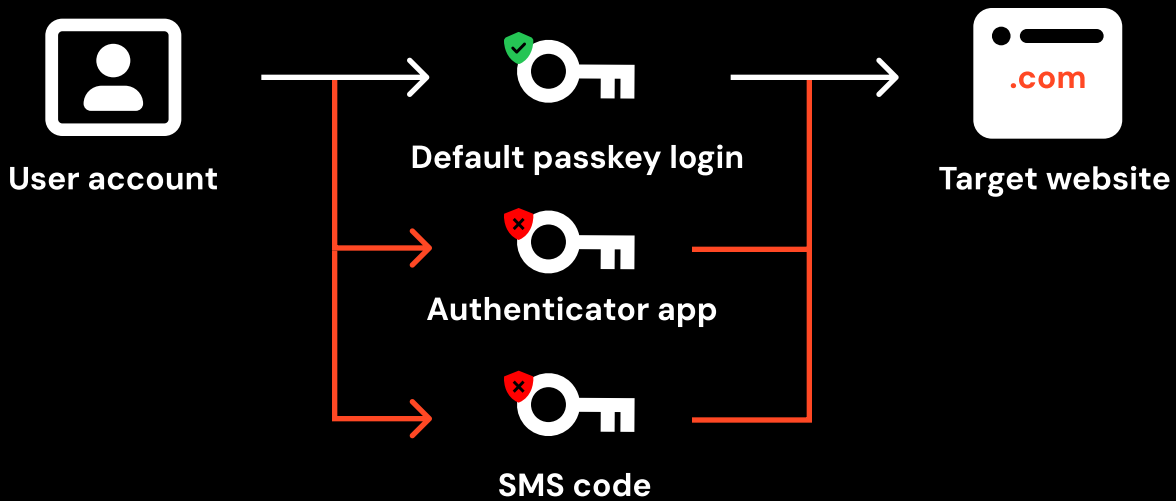
Lately though, awareness is starting to grow around some MFA methods being "phishable" (i.e. not phishing resistant). The improvement is often called a "passkey" and is typically a hardware security device that is built-into your laptop (e.g. a fingerprint sensor on a laptop) or something you plug into your device (e.g. a Yubikey). Because passkey-based logins are domain-bound, trying to use a passkey for microsoft.com on phishing.com simply won't generate the correct value to pass the authentication check, even when proxied using an AitM kit.

However, attackers have realized that even as these new phishing-resistant methods are starting to become used, most users still have alternative MFA methods active. The attacker can then do what's called a downgrade attack.

When conducting an Attacker-in-the-Middle phishing attack, the attacker doesn't need to relay 100% of the messages accurately. Instead, they can alter some of them. The app might ask the user "You need to MFA — do you want to use your passkey, or your backup authenticator code?", but the phishing website might modify this page to say "You need to MFA — use your backup authenticator code" not giving you the option to use your secure passkey.

This can also be applied to accounts that use SSO as the default login method. In this scenario, the phish kit can select a backup username and password option.

So, you have a situation where even if a phishing-resistant login method exists, the presence of a less secure backup method means the account is still vulnerable to phishing attacks. The only way to make a login 100% phishing resistant is to remove backup methods, and/or disable alternative login flows through conditional access.



Attackers can configure their phishing kits to select backup, phishable login methods.

Sandbox evasion techniques — turning legit bot protection against the good guys

The challenge for attackers when using these proxy-based phishing techniques is that it's even easier to detect in theory. Because security tools know what real Microsoft, Google, Okta, etc. login pages look like — they can simply **look for the real login page, but on the wrong domain**.

The key here is that sandbox analysis relies on that process being completely automated — though I'm calling it a "sandbox", it's technically the same technology (headless browsers) that's used to make "bots" that interact with websites. The security industry has been battling web-scraping bots for decades, triggering the creation of a whole "bot protection" industry. There's even a Gartner magic quadrant for this ("Web and API Protection").

What attackers cleverly realized is if security companies are effectively using bots to analyse their phishing websites, they could use legitimate "bot protection" products to counter them.

The tool of choice at the time of writing is Cloudflare Turnstile — you'll recognize it easily:

Cloudflare Turnstile should come with a trigger warning for security teams investigating modern phishing attacks.

Virtually all phishing kits of this generation we see today are hosted behind Cloudflare Turnstile or some other bot protection method (e.g. CAPTCHA).

The reality though is that requiring any user interaction will break these sandbox-type analysis techniques. Some attackers have taken this to the extreme — putting their phishing website behind legitimate Google OIDC login pages. In other words, you have to login to a real page to get to the fake AitM login page. Good luck training users not to fall for that.

Attacker-in-the-Middle is table stakes

Attackers don't even need to build these tools themselves. Perhaps the most popular AitM tool is Evilginx — written as a tool to be used by penetration testers, but, as is so often the case, has subsequently been adopted by criminal groups. A good example of this is recent activity attributed to Scattered Spider. All the techniques described so far are table stakes for these tools, and there are YouTube video tutorials showing you how to run them. **It couldn't be easier.**

Detection and response hasn't kept up

To make matters worse, there are very few security controls in place today that are detecting these tools. **Anything that relies on automated sandbox-type analysis isn't effective anymore.** Bot protection has seriously impaired email security and proactive scanning solutions.

Some detection techniques are going old school, and doing things like blocking newly registered domains — but these controls break legitimate websites. Threat intelligence feeds aren't working anymore either, because the tools are rarely kept alive for long enough for a human analyst to visit and analyze them. Instead, they get automated screenshots of Turnstile and then a domain that no longer resolves. With the advent of automated cloud-based infrastructure, and especially infrastructure as code, **the time taken to acquire a random domain, spin up an Evilginx instance, send a few emails, and pull them down again is trending to zero.**

The one technique that does still work (at least in principle) against this generation of phishing attacks is web traffic analysis. If you're able to inspect web traffic through a TLS-decrypting web proxy system, and you look for web traffic that matches signatures for Microsoft, Google, or Okta login pages, then that is still a viable detection method.

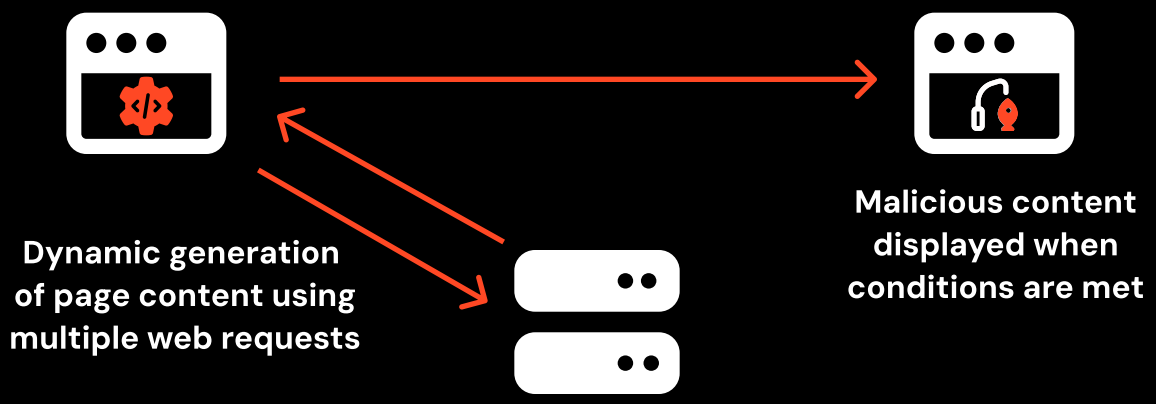
Having said that, in practice, we see many of these attacks working against targets that are protected behind proxy solutions that promise to do this kind of detection.

This could mean that the cost of analyzing every website is just too high and other domain based detections are still being relied on to trigger a deeper analysis. But perhaps a more likely explanation is that phishing tools are actually starting to use "Gen 3" techniques.

Gen 3: Modern phishing using professionalized Attacker-in-the-Middle platforms

The majority of successful phishing attacks today fall into the category of what we're calling Gen 3 attacks. It's perhaps a mistake to call these Gen 3. Compared to the leap from Gen 1 to Gen 2, these are perhaps closer to Gen 2.5, but you can be the judge!

Part of these attacks is that they are not self-contained web-proxy-like tools, but complex, customized (and customizable) dynamic web kits. These web applications load themselves dynamically using highly obfuscated JavaScript code stitched together from multiple web requests, using a changing key for each request. The decryption keys can be retrieved, but that's not the point — by doing this the pages are very difficult to analyze, and from a network perspective almost impossible to signature and therefore detect in real time.



Many detections rely on inspecting the page code to find known-bad indicators. If the page code is heavily obfuscated, these signatures fail.

Often these phishing kits are delivered as-a-Service in what appears to be more of an affiliate model. The core crimeware provider hosts a backend cloud service that interacts with the targeted websites (most commonly Microsoft, then Okta and Google).

The affiliate's job is to register domains, deploy the frontend phishing sites - which together with the backend service effectively operate in the same way — but where something like Evilginx operates at the HTTP-layer, you can think of these tools as being more of a thick-client in the middle operating more at the application layer.

All the page code, cookies, and even the rendered sites are completely custom and resemble the original, but the backend still communicates with the real login pages to pass messages back and forth. These phishing kits retain the core Gen 2 techniques (MFA bypass using AitM, and sandbox anti-analysis using bot protection) but with highly customized implementation, making detection way harder.

The main job left for the affiliate is delivering the lures and getting targets to visit these phishing sites — which is probably why we've seen so much recent innovation in this space.

Phishing delivery techniques are evolving

Likely driven more by the spam problem than phishing attacks, email providers have started to filter mass emails from unknown domains or addresses, and after decades of training to not trust emails, employees are not responding to (or even seeing) phishing emails at the same levels anymore.

To counter this, attackers are hijacking trusted sources to deliver malicious links camouflaged behind different sites. This means that even if an email is received, the email is completely legitimate — it's what comes after once the victim has left the email app/webpage. They're also abandoning email altogether in favor of alternative channels where security visibility is nonexistent.

Link delivery mechanisms

- Email from legitimate app/service
- In-app phishing
- Malvertising
- Instant messenger
- Social media
- SMS
- AI/LLM poisoning
- QR codes

Link camouflage techniques

- Email from legitimate app/service
- Trusted website hosting
- URL obfuscation

Check out the phishing detection evasion techniques matrix for more examples of these methods in action.

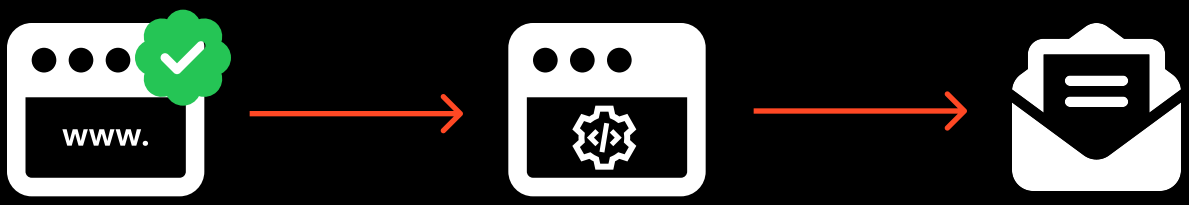
Using trusted websites to host phishing links — and deliver phishing messages

Attackers are increasingly using trusted websites to host phishing content. Any service that allows you to host a page or share a document publicly can be used to do this.

Often, these apps don't just host the document or page that includes your link (to the phishing website), it will even send your phishing message to the victim — through in-app notifications or by sending an automated email to the recipient via a third-party service.

Using trusted hosting providers and services reduces the chance of a URL being flagged by link analysis tools, while automated emails sent directly from third-party services are often expected and legitimate. This means the attacker doesn't have to invest any time or effort into building up their email reputation — they can just piggyback on a trusted domain.

This technique is similar to "Living of the Land" binaries (LOLBins) used by attackers in the era of endpoint attacks.



Attacker hosts phishing site on trusted domain, e.g.

- Azure Front Door
- Google Sites
- Cloudflare Pages
- Amazon S3
- Linode Object Storage

Attacker hosts malicious link on third-party service, e.g.

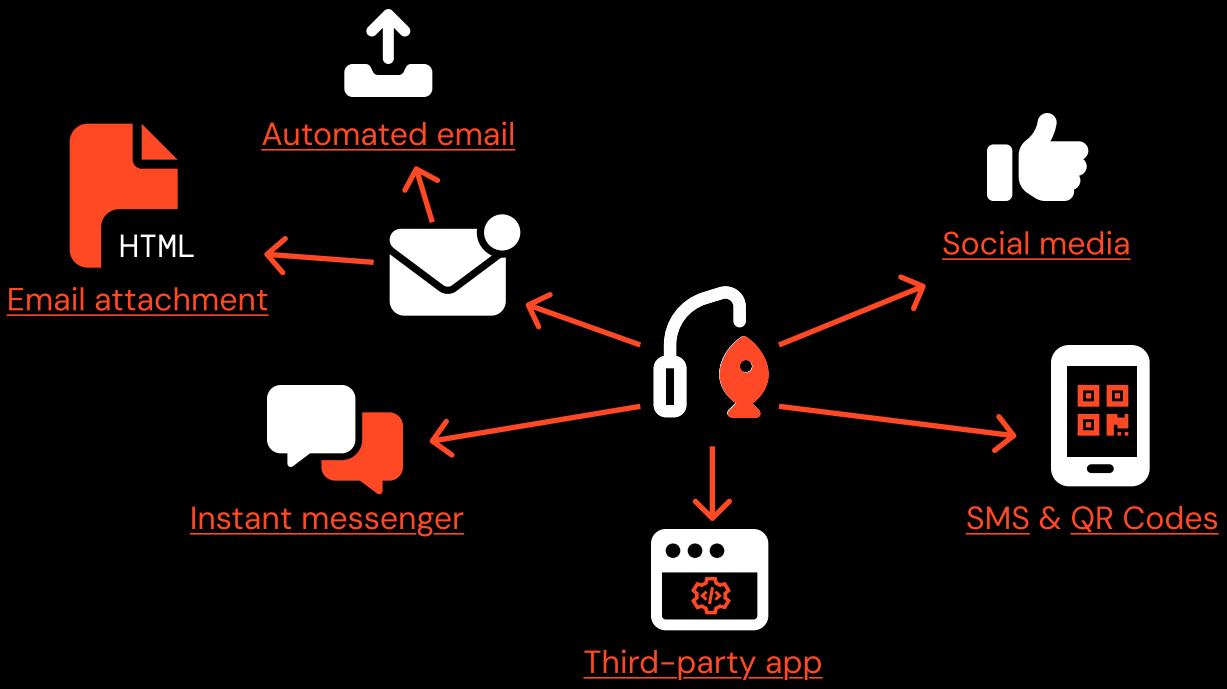
- Google Forms
- Google Sites
- GitHub
- DocuSign
- SharePoint
- Adobe

Automated email delivered from third-party service

Attackers are combining trusted domains with third-party app delivery to bypass email security filters.

Delivering links via non-email channels

Delivering phishing through email has been the de facto delivery technique for so long, we can forget that many other options exist. One thing is for sure — the old days of putting a phishing URL directly into an email without some form of camouflage or obfuscation are gone.



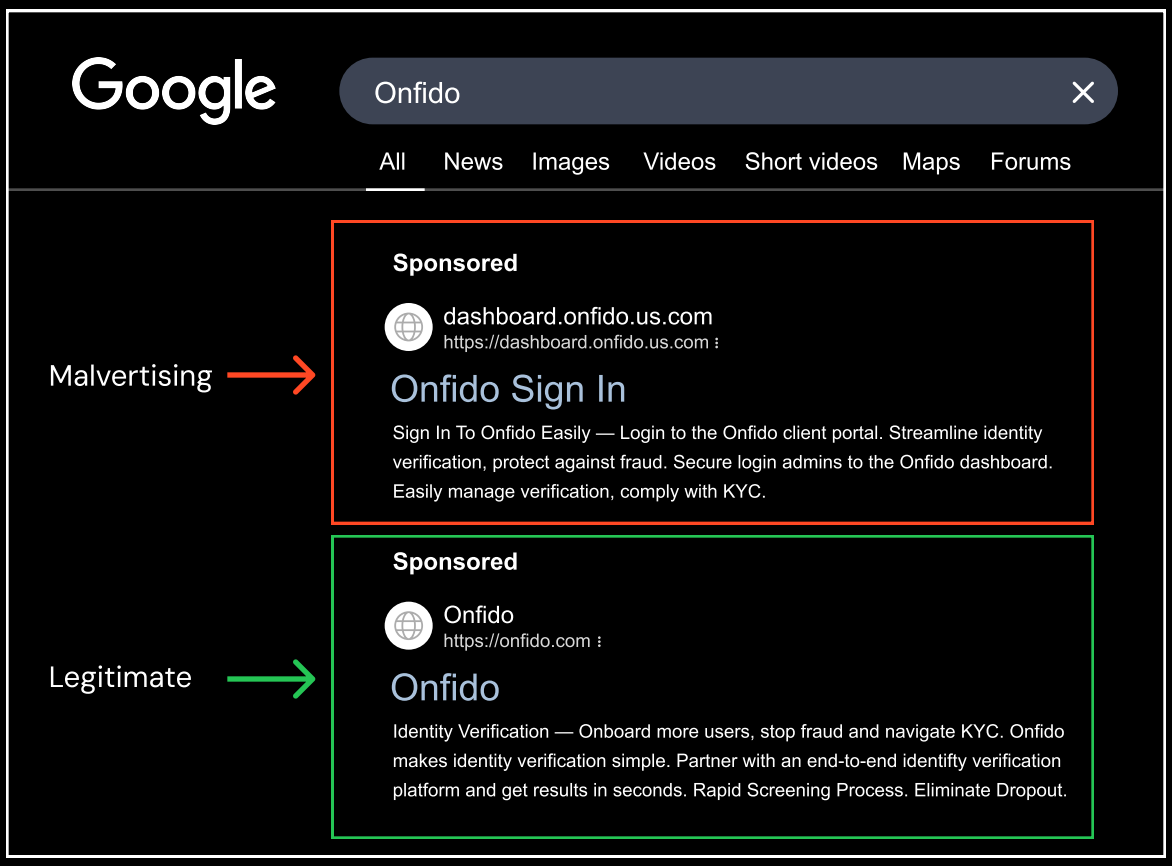
Phishing delivery channels are expanding.

As businesses shift to new communication channels like IM apps (e.g. Slack and Teams), a new vector for phishing is opening up. This makes complete sense. If Slack is where employees work, then that's the best place to target them, and with the drive to make Slack (and Teams) a more public communications platform, this is becoming easier than ever.

Other useful channels include social media apps like LinkedIn, Facebook/Meta, even Reddit. **Wherever someone you don't know can send you a message, you should expect to receive phishing lures there.**

Malvertising is another common technique, where attackers distribute malicious links via paid search engine ads or SEO poisoning (though admittedly it's harder to get your malicious content ranked on SEO than a paid ad). This takes advantage of the fact that many employees continue to search for the business apps they use every day rather than bookmark them. The malvertising link appears to be legitimate, and is usually ranked first — unsuspecting users that aren't paying attention to the "sponsored" caption will be easily fooled.

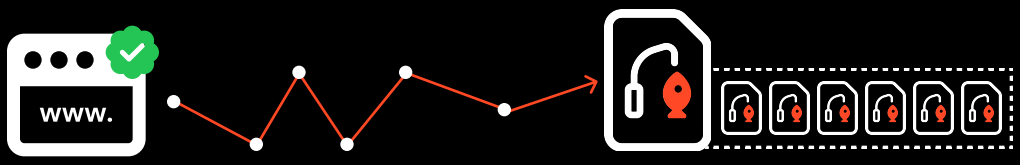
We recently intercepted an example that we later confirmed to be linked to Scattered Spider activity. In this case, the attackers were impersonating Onfido, a digital identity verification service.



Malvertising example intercepted by Push.

Further evading URL-based detections

It's worth noting a couple of additional techniques attackers are using to further prolong their phishing campaigns. Generally, attackers expect their phishing sites to have a limited lifespan. But they're extracting the most out of them by acquiring domains on an industrial scale, and using domain rotation, redirection, and load balancing to both protect the initial URLs being seeded out to victims from being flagged, and serving different phishing pages to victims — continually refreshing them from a pool of URLs.



Using redirect chains (via trusted sites) to protect initial URLs

Using load balancing to rotate the phishing page URL served

Anti-analysis and obfuscation techniques

A large part of the reason to put in the effort of creating and maintaining a fully custom phishing kit is that it's possible to significantly increase the difficulty of analysing and detecting these kits using traditional detection tools.

Most modern phishing kits make full use of this fact and implement a range of techniques to frustrate analysts, sandboxes, and detection controls.

We've already covered a few of the tricks attackers are using — namely, code obfuscation, bot protection, and domain rotation, redirection, and load balancing, but let's take a closer look at some of the more creative ones we haven't mentioned yet.

Anti-analysis

- Domain rotation, redirection, & load balancing
- Bot protection
- Legitimate OIDC logins
- Delayed execution
- Single-use links
- Conditional loading
- Anti-sandbox

Page obfuscation

- DOM obfuscation
- Page obfuscation
- Code obfuscation
- Visual obfuscation
- Desktop control & streaming
- Cross-domain iframes

Check out the phishing detection evasion techniques matrix for more examples of these methods in action.

Conditional loading & anti-sandbox countermeasures

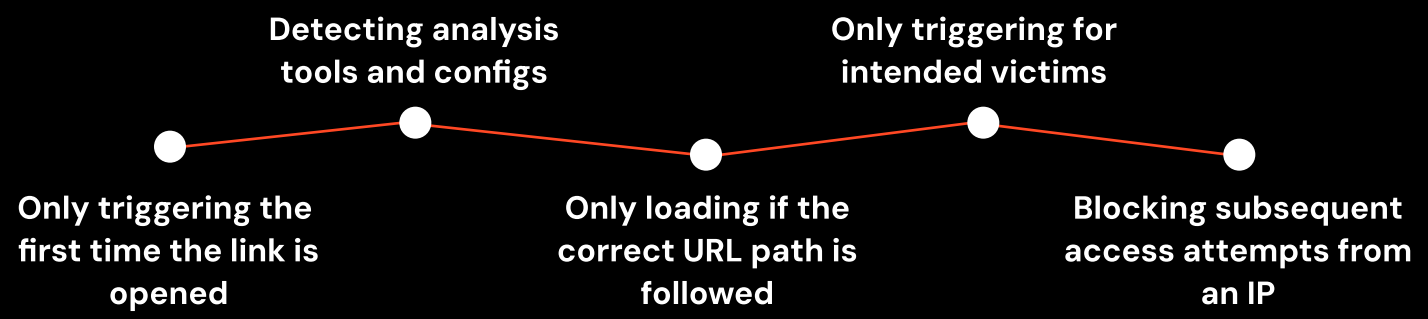
To prevent analysis of their phishing pages (and consequently, the page being flagged as malicious) attackers are using multiple techniques to block both human analysts and bots.

For example, attackers are using anti-sandbox measures to check for indicators that the page is being accessed by a security tool or analyst — looking for evidence of debuggers, running inside a VM, or analysis tools running in the browser environment. Some of these are a minor inconvenience, like adding debug statements into page scripts causing breakpoints to fire and the page to stop loading when a visitor opens dev tools to inspect the app behavior (or even disabling shortcuts to open dev tools in the first place).

Others are far trickier to get around, using conditional loading to only serve the correct (malicious) web page when the user accesses the page from the correct location, based on geo-IP. This makes the site load a benign random page if you e.g. visit it from a Madrid IP address, but a malicious phishing page if visited from Barcelona. Attackers have also been seen redirecting to a benign page if you don't provide a pre-defined target email address/domain, or if you don't follow a specific path to the URL to prevent a page being directly loaded outside of the phishing campaign. Evilginx is probably the most common example of this, where unwanted visitors are "rick rolled".

Other even more nefarious techniques are less common, like including a single-use token in the link URL that loads a phishing kit the first time you visit it, but redirects you somewhere benign if you attempt to visit it again. In this case you have one chance to view the page source and detect it — no parallel or after-the-fact analysis is possible, you have to be there when it happens the first time or you'll miss your chance.

This is crippling for sandbox-based detections that run in response to a user opening a re-written link in an email.



Anti-analysis techniques used by phishing kits.

DOM, page, and code obfuscation

If a page can be loaded by a security tool or analyst, the next step is to inspect the page elements to detect malicious or anomalous content.

Since the vast majority of phishing pages impersonate a Microsoft, Google, or Okta login page, web proxies started to search for matches with the genuine source code for these pages, but hosted on the wrong domain (i.e. a phishing site impersonating that login page).

To counter this, attackers are randomizing every part of the data that is being analyzed in the proxy or network traffic analysis tool — i.e. the DOM and web page content.

Rather than loading a complicated HTML page, then loading some JS components to make the page reactive, these kits often use a very simple "loader" HTML page. This HTML might not even contain a <html> or <head> and <body> tags, but a single script tag that loads obfuscated JS, which in turn replaces the page's DOM and then proceeds to build the page dynamically.

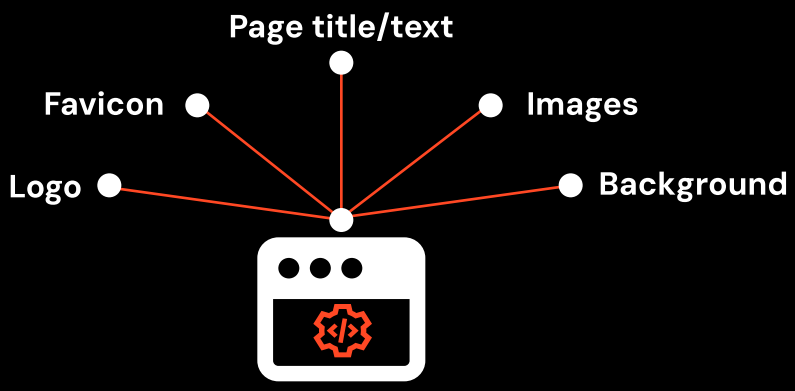
Page code is often obfuscated using encryption libraries or simple XOR encryption. The code must include the decryption keys so that it can load in the browser. But without analysing or running the code, the web payloads look completely random from a network level — making it extremely tough to detect in that way as there are no static signatures.

Visual detection evasion

Once a sandbox loads a web site, it must identify whether it is a phishing attempt. One way to do this is to take a screenshot of the browser window and run the rendered page through a computer vision model to compare it to a known or often targeted login page. **In other words, if it looks like the Microsoft login page, but isn't on a Microsoft domain, it's probably phishing.**

This technique is particularly effective at detecting reverse proxy tools that don't implement bot protection, and so some of those tools are using visual obfuscation techniques to inject scripts that subtly change the page. This might be applying an overlay blur, or a subtle color shift that is not noticeable by a human, but enough to throw off an automated comparison.

Part of the benefit with a kit that implements a custom frontend is that **you can modify and customize the page you display almost infinitely.** Practically this results in templates for commonly phished pages that look similar enough to the originals, but where the layout, backgrounds, logos, and colors are changed enough so as not to trigger detections based on real page matching.



Phish kits are dynamically generating frontend elements to defeat visual detections.

The result is that modern, sophisticated phishing campaigns are routinely evading established detection controls. This is driving the classic arms race between attackers and defenders, but one in which it's increasingly painful for defenders to keep up with the tools they have available, and attackers are always two steps ahead.

New observations and future trends

Attacker innovation isn't slowing down, and we're already seeing further detection evasion improvements in phishing kits.

Using cross-domain iframes to block injected detections

Recently, phishing kits have been seen loading the core of the page inside a cross-domain iframe. At a glance this extra step doesn't seem to provide much additional network obfuscation, but one possible reason is to protect against web-proxy based detections that function by injecting JS <scripts> to run detections once web pages actually load in the browser — a way of getting around the shortcomings of network-based detections.

Cross-domain iframes are useful in this case because the injected scripts run in unprivileged context (like any other normal script) and therefore are prevented from accessing what happens inside the iframe — where the malicious code lives. **Controls that run in the browser in a privileged position (like an extension) are needed to circumvent these protections.**

Widening the net: Expanded app targeting

Attackers have also realized how much valuable data exists in Shadow SaaS as attacks against apps like **Snowflake**, and have begun to **target apps like Slack, Mailchimp, Postman, GitHub, and other commonly-used business apps directly** — bypassing Identity Provider (IdP) accounts (MS, Google, Okta, etc.) that typically have more robust authentication and access controls in place.

The long tail of SaaS services outside of the most well-known enterprise cloud apps are often an easier target because they:

- Often have fewer security configuration options available — from not supporting SSO (or specifically SAML) to offering weaker login method enforcement (e.g. allowing several login methods to be active at once by default, creating the scope for ghost logins.
- SSO onboarding typically happens over a longer period, on an app-by-app basis. It could take years for many enterprises to formally review, risk assess, and onboard every app that employees have self-adopted and are already using.

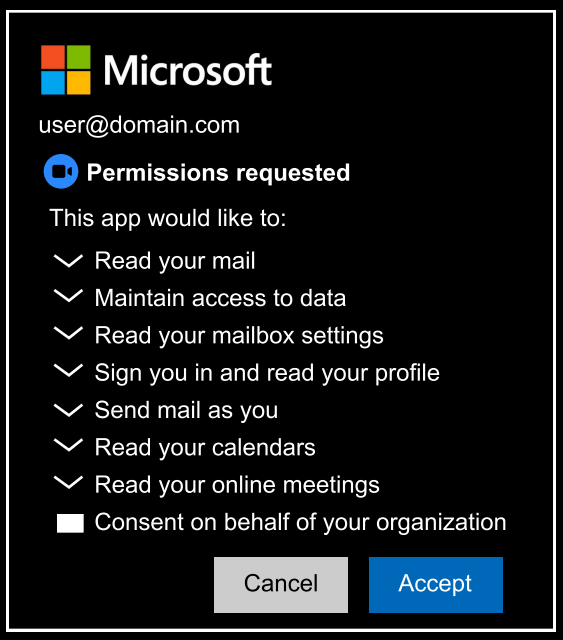
That said, attackers will continue to target IdP accounts and enterprise cloud as the targets with the highest possible payoff in terms of the potential blast radius of a compromise.

Consent phishing and other ways around "phishing-resistant" authentication

As passkeys continue to be more widely deployed and IdPs are increasingly hardened, attacks are not only moving to non-SSO accounts, but they are also shifting to attacks that circumvent the standard authentication process entirely.

OAuth consent phishing involves tricking a user into approving a malicious OAuth integration — effectively giving the attacker a programmatic token that can be used to access their account.

This was one of the first techniques we added to the SaaS attacks matrix and has been known for the better part of a decade. Despite this, it's not yet become widely adopted at anything like the scale of credential phishing attacks, but may well do so when passkeys and other access controls become effective and widespread enough to put a dent in typical phishing attacks.



If you're interested in other emerging techniques like consent phishing, you can also read more about device code phishing, verification phishing, and app-specific password phishing.

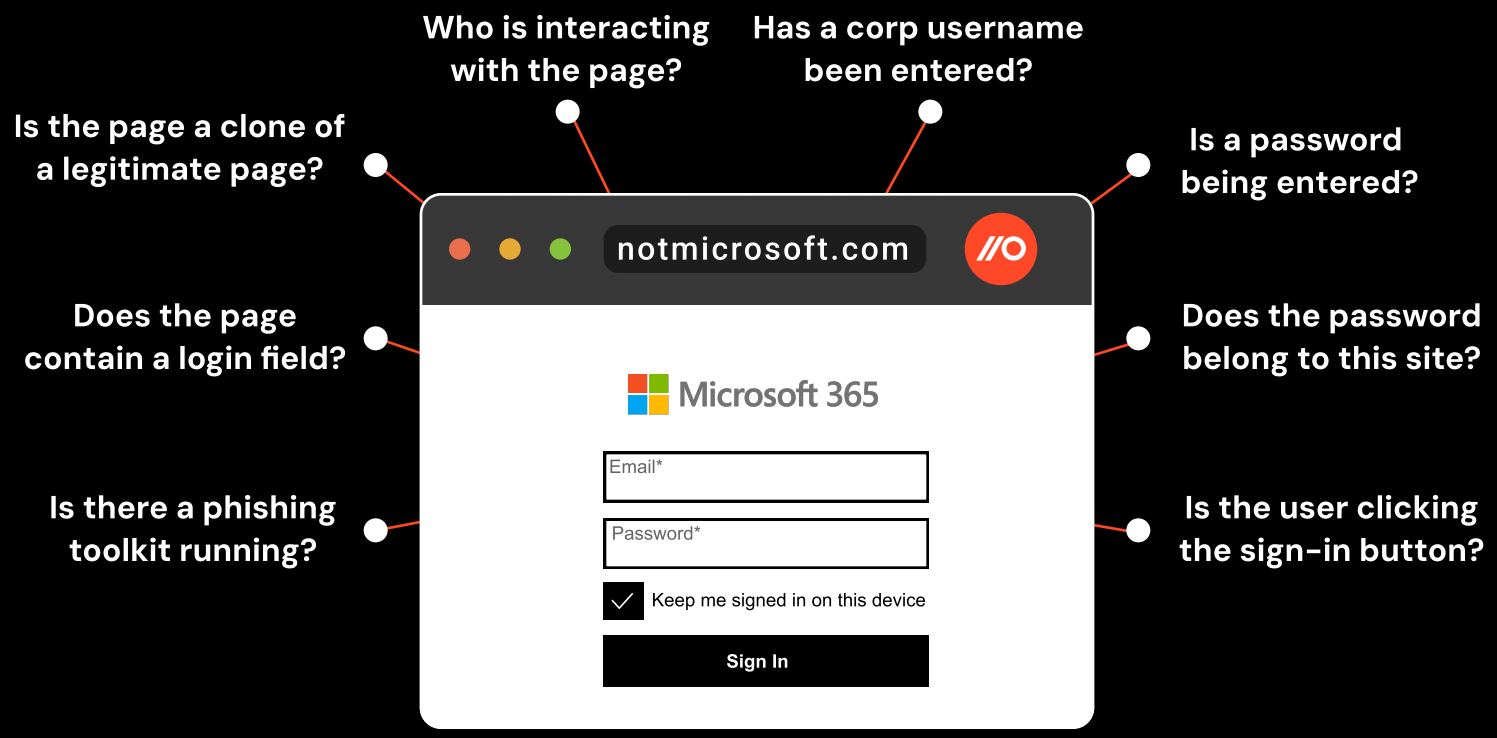
One thing is sure — expect attackers to adapt and find new bypasses as the control environment changes.

Push is like EDR, but for your browser

At Push, we use a browser agent to detect, intercept, and shut down phishing attacks in real-time, as they happen in the user's browser.

Push changes where phishing protection happens, from upstream detection to point-of-interaction control. Instead of chasing malicious links through email gateways or external threat feeds, Push embeds lightweight, always-on protection directly, as users go about their work in the browser.

Push monitors what's happening in each session: how pages are built, how they behave, and how users interact with them. That means it can recognize when a login prompt doesn't match your identity provider, or when a script behaves like part of a phishing toolkit.



Push provides deep browser context to detect and block phishing attacks.

Push tackles these attacks using a new type of telemetry. We don't look at emails, we don't look at URLs or proxy logs, there's no list of bad domains or TI feeds. Push provides protection at the last mile to stop even the most sophisticated attacks and advanced phishing kits.

No matter what the delivery vector is, all roads eventually lead the user to a malicious page, accessed in their web browser. The browser is where page rendering meets user input, code execution and network traffic — and therefore browser security agents present the most valuable opportunity to detect and block current and future generations of attacks.

No other solution takes the power of the browser and combines it with identity data, fine-grained browser telemetry, and research-driven detection engineering to stop attacks.

It's not just advanced phishing attacks either. Push's browser-based platform protects against credential stuffing, password spraying and session hijacking using stolen session tokens. You can also use Push to find and fix identity vulnerabilities across every app that your employees use, like: ghost logins; SSO coverage gaps; MFA gaps; weak, breached and reused passwords; risky OAuth integrations; and more.

Book a demo at pushsecurity.com to see Push's browser and identity security features in action, and watch a phishing attack get shut down in real time.