Trap-Aware Client-Side Email Reveal: Runtime Obfuscation, Behavioral Gating, and Honeypot Attribution

Author Name(s) Withheld for Review
Institution / Organization
City, Country
contact@example.org

Abstract

Public email addresses remain useful contact points on websites, but they also create low-cost targets for automated harvesting. Existing client-side defenses typically address only one part of the problem: static concealment, explicit friction, or coarse signaling. This paper presents a compact version of a trap-aware client-side email reveal framework. The framework reframes email exposure as controlled disclosure with parser-method attribution. It combines split-share runtime obfuscation, a behavioral reveal gate based on trusted interaction signals, environment-aware policy checks, a structured honeypot attribution matrix, and a post-reveal output policy for copy, `mailto`, and selection channels. The system is a client-side friction and attribution layer, not a cryptographic security mechanism. Its goal is to raise extraction cost, preserve legitimate access, and produce probabilistic attribution signals that become operationally useful when correlated with mailbox or server-side telemetry.

CCS Concepts

• **Security and privacy** → **Browser security**; *Usability in security and privacy*.

Keywords

email protection, web scraping, honeypots, behavioral gating, browser automation, Shadow DOM

1 Introduction

Public-facing sites often expose email addresses because email is interoperable, accessible, and compatible with existing organizational workflows. The same properties make addresses attractive to automated harvesting systems. Empirical work has shown that addresses published on web pages may be collected and later used for unsolicited mail and abuse measurement [1, 2]. Modern automation further broadens the threat surface: a collector may read raw HTML, match regular expressions, execute JavaScript, wait for rendering, dispatch synthetic events, or extract post-reveal output through copy, `mailto`, or selection channels [3, 4].

The design problem is therefore not simply whether an address is hidden or shown. A useful public contact channel must remain usable for legitimate users while making automated collection less silent and less uniform. Static obfuscation and JavaScript reconstruction can delay plaintext exposure, but once an address is recovered they usually provide little evidence about how extraction occurred. CAPTCHA-style gates add stronger friction but are disproportionate for lightweight contact reveal and raise well-known usability and accessibility concerns [5–7]. Generic honeypots can reveal that collection happened, but a single decoy rarely distinguishes source parsing from synthetic activation or post-reveal scripted output.

This work proposes trap-aware client-side email reveal: a framework for controlled disclosure with structured attribution. Its goal is not complete resistance to automated extraction. Instead, it raises cost, preserves legitimate access, and classifies suspicious extraction paths with probabilistic signals.

2 Framework Overview

The framework has five layers. First, *split-share runtime obfuscation* ensures that the complete real address is not present as a ready plaintext string in initial HTML, pre-reveal DOM, `textContent`, or `mailto` attributes. Address material is distributed across HTML and CSS or JavaScript shares; local-part and domain are handled separately; and reconstruction occurs late, after policy approval.

Second, a *behavioral reveal gate* requires signals consistent with legitimate interaction: pointer/touch press chain, bounded timing, movement cancellation, keyboard accessibility paths, `Event.isTrusted`, optional `navigator.userActivation`, and a bounded Web Worker friction step. The Worker is a friction layer, not cryptography. Third, *environment scoring* considers signals such as `navigator.webdriver`, pointer/hover coherence, viewport behavior, and storage availability. These are policy inputs, not proof of identity.

Fourth, the system uses a *honeypot attribution matrix*: suspicious paths receive decoys whose local-parts encode both behavioral mode and output channel. Fifth, a *post-reveal output policy* applies the principle `visible != output`. Trusted copy, `mailto`, and selection-copy receive normalized real output; synthetic post-reveal output can be blocked, tainted, or routed to decoys. In hardened deployments, closed Shadow DOM and zero-width pollution reduce naive post-reveal DOM extraction, while remaining only hardening mechanisms rather than security boundaries [9, 10].

3 Attribution Model

The static offscreen honeypot is treated as a separate static channel, conceptually `infoweb@...`. It captures primitive source, `mailto`, and regex collectors. Dynamic attribution uses a mode-by-channel matrix. Core modes include `trusted_false`, `missing_gesture`, and `up2click_slow`; higher deployment levels can add `up2click_fast`, `env_suspicious`, `post_reveal_synthetic`, `retry_abuse`, telemetry-only cells, and per-session aliases.

This matrix is the target architecture. A reference implementation may activate only a subset of cells, while telemetry and adaptive deployments can enable richer channel routing. The important distinction is that a decoy address is not merely a trap; it is a compact hypothesis about the extraction path. Full operational value requires mailbox or server-side correlation.

Table 1: Compact attribution examples.

Mode / signal	Response	Attribution meaning
Static HTML / `mailto`	Static `infoweb` decoy	Source-level or regex collection
`isTrusted=false`	`hp_tf*` decoy	Synthetic JavaScript activation
Missing press chain	`hp_mg*` or retry	Incomplete gesture automation
Slow up-to-click	`hp_slow*` or telemetry	Timing mismatch after press-end
Synthetic copy / selection	`hp_prcopy`, `hp_prsel`, or block	Post-reveal scripted output
Environment anomaly	Keep masked or telemetry	Incoherent browser context

4 Reference Implementation

A reference implementation organizes the runtime around explicit contracts: obfuscated HTML attributes, CSS/JavaScript shares, late address reconstruction, a honeypot address builder, a centralized policy engine, and state factorization for gesture, Worker, keyboard, output, and session concerns. The framework-level state vocabulary includes `MASKED`, `GESTURE_TRACKING`, `PENDING_WORKER`, `REVEAL_DECISION`, `REVEALED_REAL`, `HP_RETRY`, `HP_LOCKED`, `HP_QUARANTINE`, `KEEP_MASKED`, and `TAINTED`. An implementation may map these abstractions to more granular identifiers such as separate pending Worker states, cancellation states, policy decisions, and taint flags.

Production deployment should strip debug-only helpers, preserve runtime helpers needed by the release build, run smoke tests, and generate integrity artifacts such as checksums or Subresource Integrity metadata where applicable. Deployment can be progressive: minimal concealment, behavioral gating, attribution routing, hardened post-reveal output, telemetry correlation, and adaptive per-session aliasing.

5 Evaluation and Demonstration

Evaluation should measure four properties: pre-reveal concealment, interaction friction, attribution quality, and false-positive behavior. Static HTML, regex, and DOM-before-reveal tests should not obtain the real address. JavaScript-aware static analysis should require combining multiple resources and runtime context. Synthetic `element.click()` and `dispatchEvent()` tests should naturally enter suspicious paths because such events are not trusted. Legitimate pointer, touch, keyboard, copy, `mailto`, and selection-copy paths require real-input testing through manual interaction, CDP/native input, or equivalent mechanisms.

A two-page demonstration should show: static honeypot collection; successful pointer/touch/keyboard reveal; synthetic click routed to `trusted_false`; missing gesture routed to recovery or decoy; hardened post-reveal rendering; trusted selection-copy returning normalized real output; scripted selection extraction routed as suspicious output; and mailbox correlation for at least one decoy class.

6 Limitations

All client-side code and data are observable to a determined adversary. The framework does not provide cryptographic secrecy, does not establish human presence, and does not claim complete resistance to automated extraction. Native-input automation can closely imitate legitimate behavior. Environment scoring can create false positives for privacy-focused browsers, assistive technologies, kiosk devices, and unusual configurations. Honeypot addresses encode likely extraction paths, but attribution is probabilistic and should be correlated with mailbox and server telemetry.

Despite these limits, the framework moves public email protection beyond passive obfuscation. It turns reveal into a controlled, policy-governed, telemetry-ready process that preserves legitimate access while producing structured evidence about suspicious extraction paths.

References

[1] C. A. Shue et al. 2009. Spamology: A Study of Spam Origins. *CEAS*.
[2] G. Schryen. 2007. The impact that placing email addresses on the Internet has on the receipt of spam. SSRN.
[3] X. Li et al. 2021. Good Bot, Bad Bot: Characterizing Automated Browsing Activity. *IEEE S&P*. DOI:10.1109/SP40001.2021.00079.
[4] H. Jonker, B. Krumnow, and G. Vlot. 2019. Fingerprint Surface-Based Detection of Web Bot Detectors. *ESORICS*. DOI:10.1007/978-3-030-29962-0_28.
[5] E. Bursztein et al. 2010. How Good Are Humans at Solving CAPTCHAs? *IEEE S&P*. DOI:10.1109/SP.2010.31.
[6] A. Searles et al. 2023. An Empirical Study & Evaluation of Modern CAPTCHAs. *USENIX Security*.
[7] W3C. 2021. Inaccessibility of CAPTCHA. W3C Working Group Note.
[8] P. Laperdrix et al. 2020. Browser Fingerprinting: A Survey. *ACM TWEB*. DOI:10.1145/3386040.
[9] MDN Web Docs. 2025. Using shadow DOM; ShadowRoot: mode property.
[10] web.dev. 2016. Shadow DOM v1: Self-Contained Web Components.