

The Blind Spot in the Browser

What your web pages are made of, where each piece comes from, and what changes over time.

Jason Gillam
Mechanical Drake, LLC

15 November 2026

Summary

When someone visits your website, their browser does a lot more than just display a web page. It fetches code, images, fonts, styles and data from a list (sometimes a long list) of other locations, often from different companies. Some items get into the browser through a dynamic chain that gets resolved at runtime instead of being specified directly in your website code. And to make things more interesting, that list of dynamic dependencies can change from time to time without warning. This paper is about how I built WyrmSight to collect, monitor, and visualize that connection data.

If you only take three things away, I'd like them to be these:

1. A modern web page may load code from a long list of outside domains, and at least one in five of those is loaded by another company's code. On about half of sites, that list changes within a week.

2. The tools most teams already run either watch the servers and the network traffic the organization controls, or scan the code at build time. Traffic that goes straight from a visitor's browser to another company passes through neither, so this layer is a blind spot for them except during a penetration test or when someone is actively examining it in browser dev tools.

3. WyrmSight watches this layer of data from where it actually happens, in a real browser, every day, and reports what changed. This gives you insight into the application, what it connects to, and how that changes over time.

1. A page load in the browser

I have spent a lot of years doing web application penetration tests, and a big part of that job is reading proxy history in tools like Burp Suite or ZAP (i.e. a web proxy is a tool that sits between my browser and the application I am testing, and logs every request and response). Part of the work is moving around an application and watching that request/response traffic in the log. On a modern single-page app, the first page load alone populates that log with connections before I have clicked anything, and I found that I could get a lot of insight from that initial page load and its dependencies.

For a lot of organizations, a pen test is the only time anybody sees that full view. With the right tooling you can see it every day, for every application.

Like a cereal box

A box of cereal lists its ingredients. When one of those ingredients is itself a mix (e.g. cinnamon bits), it gets its own list in brackets: cinnamon bits (sugar, wheat flour, cinnamon, sunflower oil). A web page doesn't announce its ingredients anywhere you can see it easily.

If it did, the tag manager would be the cinnamon bits. A tag manager is one script a marketing team adds to a site so they can add other tools later without waiting on a code release. Whatever they configure in it is the bracketed list.

Security folks will recognize this concept from software bills of materials (SBOMs). An SBOM covers what a team builds and ships. But it typically does not cover what the visitor's browser assembles when the page loads, so a tag manager could dynamically load resources that were invisible when the SBOM was created.

One ordinary page load

Here is roughly what happens when someone opens a typical company homepage:

1. The browser fetches the page from the company's own server, along with the company's own scripts, styles and images. That is first party.

2. The page loads an analytics script, probably one the company chose, from the analytics vendor's servers. That is a third party.

3. The page loads a tag manager, which is also a third party, and also chosen.

4. The tag manager loads whatever someone has configured in it: an advertising pixel, a heat-mapping tool, a chat widget. Someone set these up in the tag manager, outside your website code. Since a third party (the tag manager) loads them, that makes them fourth-party resources. They may load pieces of their own too, and some of those come from still other companies (some you may not even have a contract with), so the chain can run to a fifth party and beyond. The important thing to know is they are not first party, not under your control, and often a surprise.

All of that happens on the visitor's machine, in the first few seconds after typing in a URL.

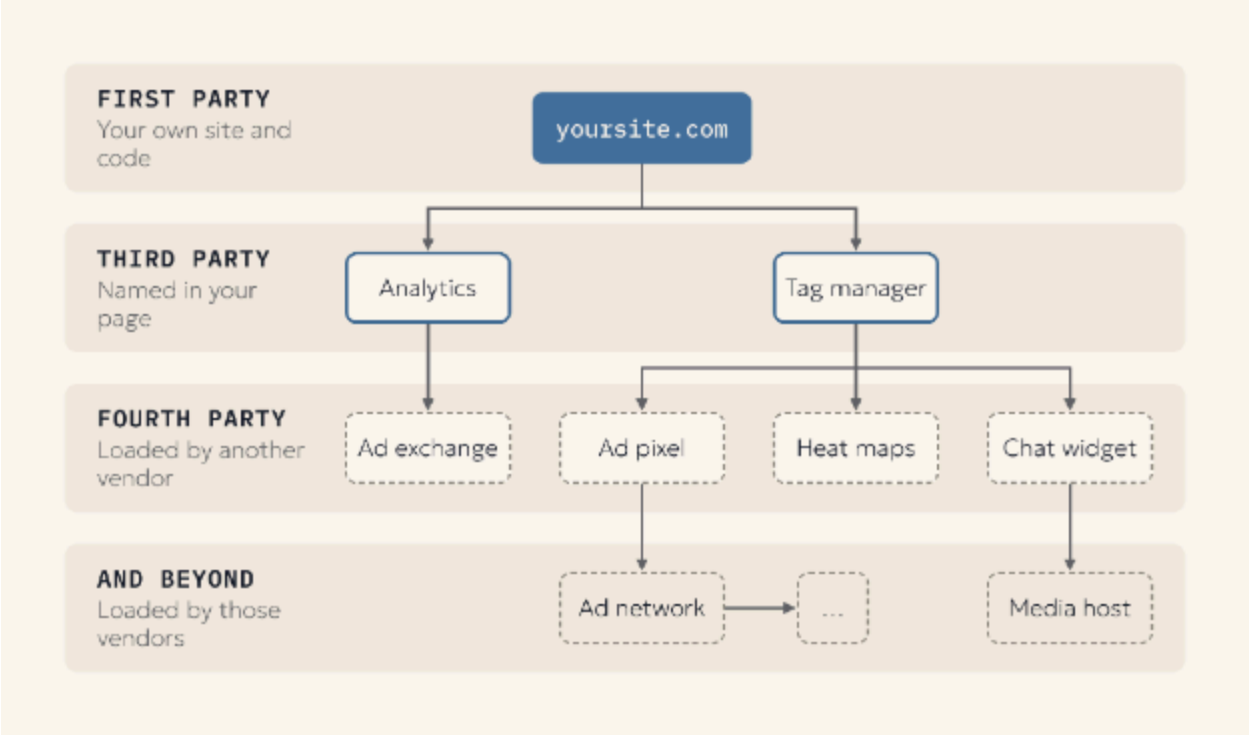

Figure 1: What one ordinary page load pulls in. Your site sits at the top, with the two vendor scripts loaded directly from the page. Everything below them was loaded by another vendor, and that lower half is what your vendor list may be missing.

To understand these patterns I have been watching 200 household-name websites every day. The numbers below come from one week in October 2026, but they are representative of the patterns I have seen for the past few months. These are the companies' public homepages and marketing sites (not their apps). Here is what stood out:

- About three in four sites (74%) load some kind of tag manager.
- On 58% of those, at least one other company's domain arrives only through the tag manager. For a typical tag-manager site that is one company, but one in ten has 12 or more (i.e. the data has a "long tail").
- Meta's pixel shows up only through a tag manager, on a quarter of the sites that use one. That's one of the most common tag-manager loads.
- At least one in five third-party domains is reached only through another third party, and up to one in three depending on how strictly you count. The longest chains run four hops from the page (i.e. your page loads a resource from one

domain, which loads another resource from a second domain, which then loads another resource from a third domain, and so on).

To be fair to the people running these sites, "only through the tag manager" means the page itself never requested anything from that vendor directly. Obviously somebody did configure the container, it's just that the choice never shows up in the site's code anywhere so it is often invisible unless you are watching browser traffic.

What a browser will show you

Every browser has a version of the same pen test web proxy data I describe above built in. Open the developer tools, click the Network tab, reload, and you get the list I am describing: every script, image, call and frame the page requested, and where each one came from. WyrmSight records the same thing. A scan is no different than a human with DevTools open.

Now put that list next to your vendor register and you have the gap this paper is about.

2. The blind spot

Most security tools watch your servers and your code. But what actually happens at runtime in the browser doesn't always match what we think is going on. That third-party layer gets assembled dynamically, and it can change frequently. Part of it arrives through other vendors' code. The people responsible for it often only see part of the overall picture.

Where the web app code runs

Server logs record what your server was asked for. A web application firewall inspects traffic coming into your site. Code scanning and dependency scanning read what is in your repository. All of those are worth having, and all of them look at your server or your source. When a visitor's browser fetches a script from a vendor's CDN, that request goes from the visitor straight to the vendor. It never touches your server, and the script was never in your repository. If the vendor changes the file tomorrow, none of those tools has anything to compare it against.

Additive risk

Let's say your team approved one tag manager. Then whoever runs the tag manager added more tags, and some of those tags load more vendors of their own. Meanwhile

the vendor register was built by asking each team what they use at build time, which might still be accurate for the build, but does not represent what is actually running in the browser.

The numbers

From the same 200 sites I mentioned earlier:

- **A typical site contacts 11 third-party domains** (that's the median). But the average is 19, and one site in ten contacts 46 or more (that is not a typo).
- **94% run JavaScript from at least one third-party domain.** That's expected. But 38% run JavaScript from ten or more third-party domains!
- **48% changed their set of third-party domains at least once** during a week of daily loads.
- **948 distinct third-party domains** showed up across the 197 sites. A handful of companies show up on many sites: Google reaches 78% of sites, Adobe 41%, Meta 35%, Microsoft 33% and OneTrust 33%.

Low priority sites

A company's marketing site may not be the product or app that it sells. That is exactly why I think all sites matter. Marketing sites, customer portals and admin panels are a low priority for most AppSec scopes and pen test schedules, but they are tied to the company, its reputation, and often some of its customer data all the same.

3. Why it matters now

Supply chain in the browser

A vendor can change a hosted script without anyone at your company shipping anything. Most of the time that is fine (it is the whole point of a hosted script). It is also how a publicly documented kind of attack works: someone compromises a third-party script that runs on checkout pages and adds a few lines that copy card details as people type them. In my experience the hard part of that problem is noticing it. I don't think any of this calls for alarm, but it is worth knowing what is there.

Are there formal or regulatory requirements for this data?

The data I am describing (what each page loaded, from whom, and when that changed) turns out to be relevant in quite a few places once you start looking. A few examples:

- **PCI DSS.** Version 4.0.1 has two requirements relevant to this, both mandatory since March 31, 2025. Requirement 6.4.3 covers every script loaded and executed on a payment page in the visitor's browser: each one must be authorized, have its integrity assured, and appear in an inventory with a written business or technical justification for why it is needed. The standard says this applies to scripts from third and fourth parties too. Requirement 11.6.1 calls for a change- and tamper-detection mechanism that alerts people to unauthorized changes to a payment page's security-impacting HTTP headers and script contents as the browser receives them, run at least weekly or at a frequency set by a targeted risk analysis.
- **Privacy.** Records such as a GDPR Article 30 record of processing start from someone's list of what the company uses. If you leave a vendor off that list then the record is wrong. Consent banners have their own gap: on 69% of the sites I watch, a third party set a cookie before the visitor had clicked the banner. (My scans never click the banner, and some banners aren't shown to US visitors at all, so this number covers only what happens on arrival.)
- **More generally,** regulators increasingly expect a company's third-party list to match reality.

4. How WyrmSight approaches it

Here are five of the big choices I made in designing WyrmSight, and why I made them:

1. Observe in a real browser

WyrmSight loads each application the way a visitor would and records every connection the page makes: scripts, requests that send or fetch data, frames, and anything those pull in. For pages behind a login, a scripted session signs in with credentials the customer provides and walks the page. Credentials are stored encrypted and used only during a scan, and how much of this you get depends on the plan.

I chose this because it records and reports what actually happened. It also means you can point it at your own production site with a clear conscience, because it does what a visitor does. WyrmSight doesn't spider the site, inject traffic, try to get around authentication or test for vulnerabilities. It doesn't block, fix or enforce anything either; I would rather the people who run a site decide what to fix. And it records what each script loads and where it connects, without analyzing what the script's code does.

Of course, all of this depends on the capture being right, and that is harder than it sounds. HTML is messy, browsers are messy, and both carry years of backward compatibility nuances. There is no simple test that proves a capture is complete. The only real test is to run it against lots of different sites and check what comes back, so I regularly test the capture of public data from a large number of brand sites.

2. Inventory first

WyrmSight lists every domain each application connects to and splits them into first party and third party, with each vendor's purpose (e.g. analytics or advertising) where it can be identified. It also identifies the common frameworks and libraries an application uses, and shows how many it could not identify. Add your whole fleet of sites and apps, and you get one picture of the browser connections across all of them.

I put the inventory first because you need a baseline before you can spot what changes.

3. Changes

WyrmSight scans every application daily, and you can run a scan whenever you want. Every connection and third-party script carries a dated history: when it first appeared, when it changed, and when it went away. A change feed lists what moved since you last looked.

A full inventory on its own is a burden. Hundreds of connections across your applications means hundreds of things to read. Since WyrmSight watches all of them every day, you can review changes and match that up with your change management process to make sure everything was expected. This approach combined with some filtering helps keep even larger datasets manageable.

4. Four risk areas

I took a lot of time deciding on a way to measure risk. I didn't want it to look like a vulnerability scan, where people are used to seeing a CVSS score, so I decided each application gets a risk profile across four areas: Supply chain, Exfiltration, CVEs and Integrity. Each area stands on its own, and the catalogue of the checks in each area is published in WyrmSight for reference. Exfiltration covers where data goes. It does not recognize personal data or secrets in what a page sends (that is a future direction though, see section 6). WyrmSight flags library versions with published advisories, and it flags connections to domains on threat-intelligence lists.

For a sense of scale on the library side: on at least 26% of the sites I watch, the page exposes a front-end library version with a published advisory. That is a floor, since 55% of those sites expose no readable version at all. And a published advisory does not mean the site is exploitable; many advisories apply only if the page uses a particular feature of the library.

5. Exportable records

I decided to make all the main lists in WyrmSight exportable as CSV. If you have a business case for exporting something else, let me know and I will consider it. I can't predict what question someone will have about a page later on. It might be an internal review, a privacy question, or an auditor asking what a page loaded. If the record is still there, or you exported it, someone can go and look it up. What that data means for your obligations is your call. WyrmSight doesn't produce compliance reports, attestations or evidence packs at this time.

Why CSP keeps coming up

There is a second reason this product exists. When I started researching Content Security Policy (CSP) patterns a few years ago, I found that it covers far more than which scripts may run. Almost anything that touches the same-origin policy or a third-party connection has a directive that can lock it down. Point a strict policy in report-only mode at most sites and the violation log fills up fast.

So WyrmSight drafts a starting CSP from what it observed on your pages. It is basic, and it is a starting point. Whether and how you enforce it is up to you.

Day to day

- An alerts list, an email when something new needs attention, and a digest. When you have reviewed an alert, you can accept it.
- Where connections are located, by country, as far as an IP location lookup can tell. You list the countries you allow (the EU and EEA are available as groups), and a connection located anywhere else raises an alert. Some connections can't be placed in any country, and those raise no alert, so treat it as a useful signal with gaps.
- Teams with invitations and roles, an audit log, and multi-factor sign-in.

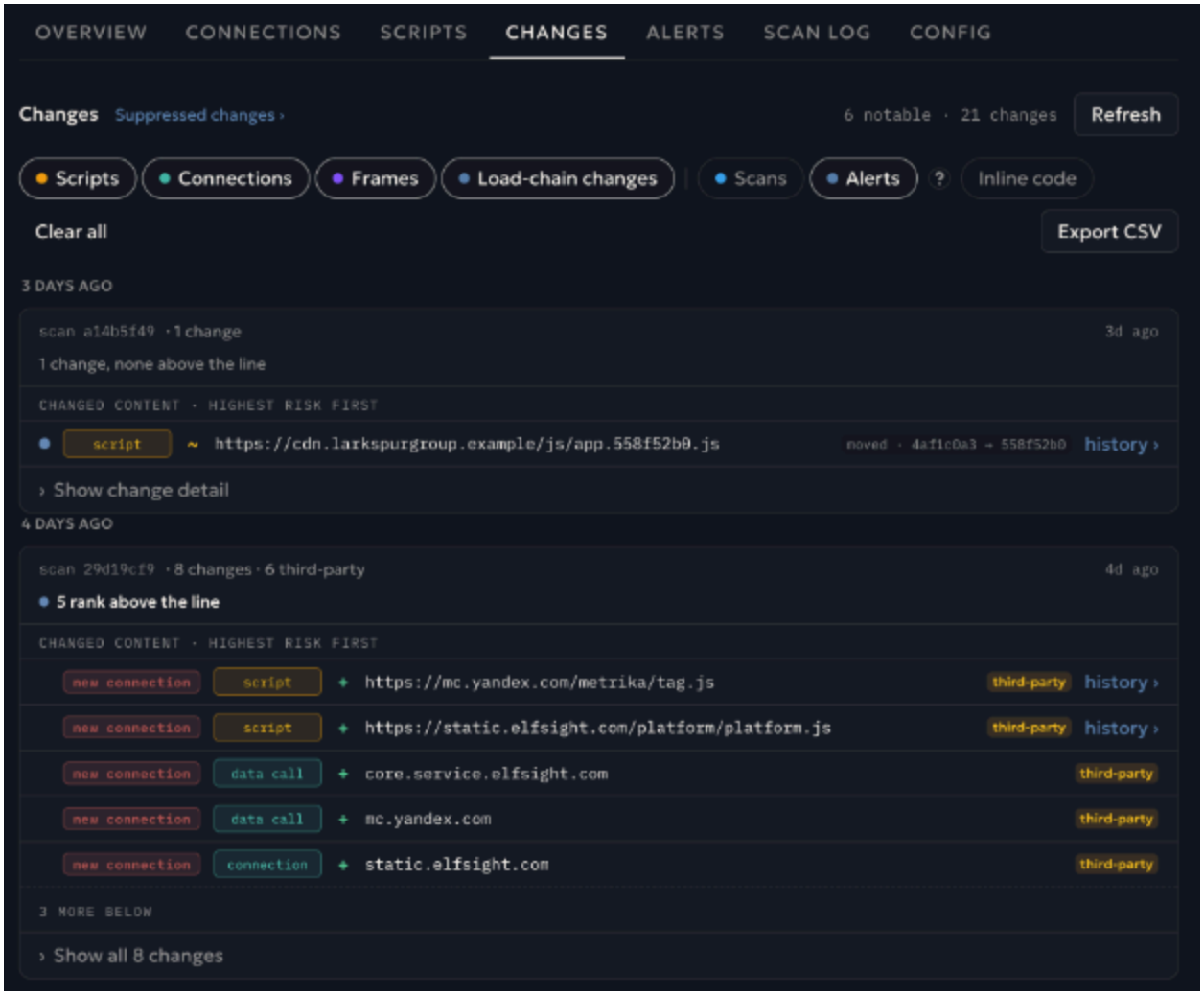

Figure 2: The change feed in WyrmSight, scanning a set of demo sites I built for screenshots so that no real company's site appears. Changes are grouped by scan, newest first. Here one scan found a script that changed, and the scan the day before found eight changes, the top five being new third-party connections.

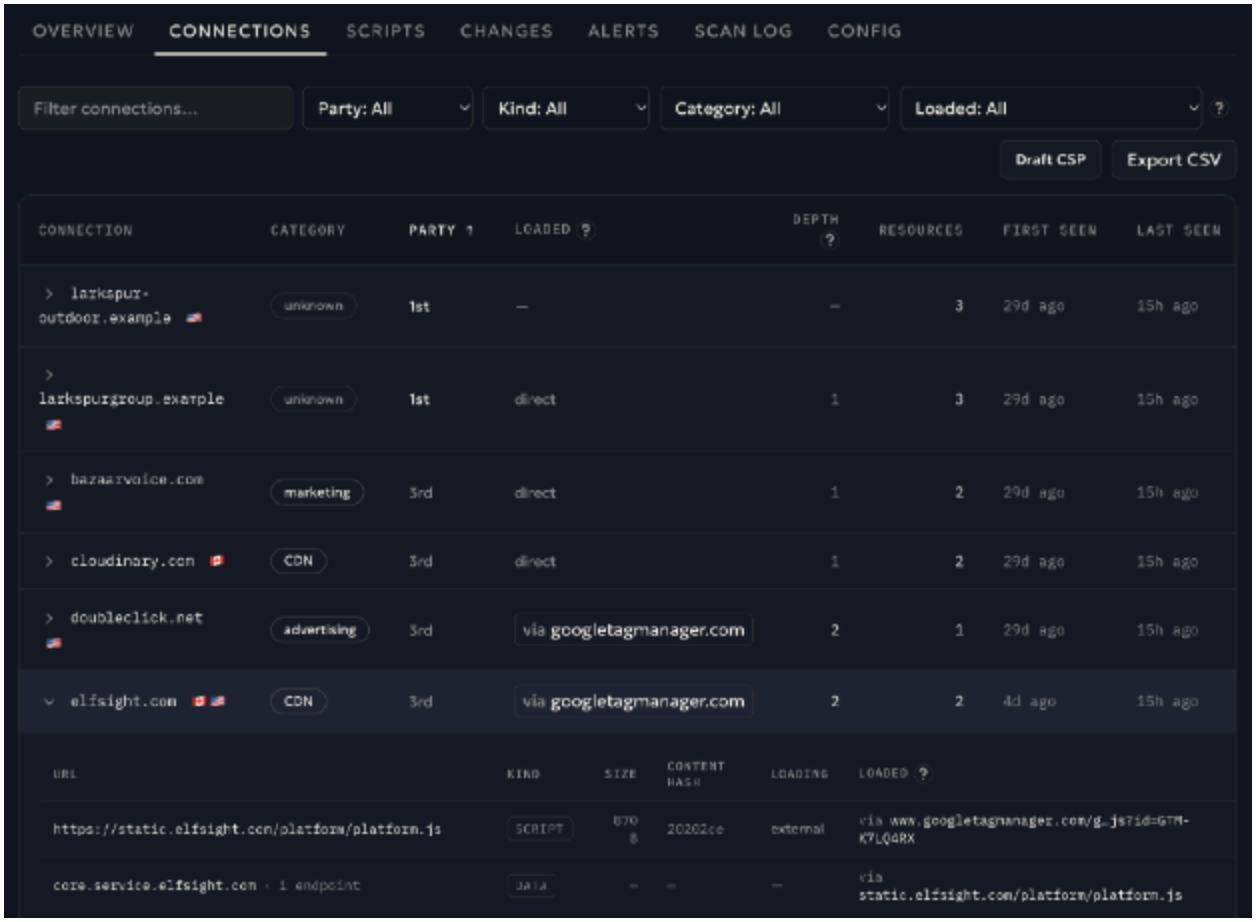

Figure 3: The connections inventory, from the same demo sites. Shows the domains each application talks to, split into first and third party, with vendor purpose where identified. Party and Depth are two separate measures: Party says who owns a domain, and Depth counts the fewest hops from the page. A third party that another vendor loaded (doubleclick.net and elfsight.com here, both via the tag manager) is what this paper calls a fourth party. The expanded row shows a chain: the tag manager loaded Elfsight's script, which then called Elfsight's own data service.

5. Try it on your own site

You can see the start of this yourself in about five minutes, with nothing but a browser.

1. Open your homepage in Chrome, Edge or Firefox, open the developer tools (F12 on most systems), and click the Network tab.

2. Reload the page. If there is no Domain column, right-click a column header and add it, then sort by it.

3. Write down every domain that is not yours.

4. Put that list next to your vendor register and see how many rows match.

5. Do it again next week and compare.

For practitioners, there is a second version that uses the browser's own reporting. Deploy a strict Content Security Policy in report-only mode, something like this:

```
Reporting-Endpoints: csp-endpoint="https://example.com/csp-reports"
Content-Security-Policy-Report-Only: default-src 'self'; report-uri
https://example.com/csp-reports; report-to csp-endpoint
```

Swap both example.com addresses for one on your own site that can collect the reports (browsers differ in which of the two they use, so set both). Report-only means the browser won't block anything. It just sends a report each time the policy *would* have blocked something. Then watch the reports arrive.

None of these techniques is new. I have used a browser's network log, a proxy history and CSP reports for years. The real effort goes into building a consolidated dataset across all your applications, every day, and keeping the history so you can see what changed. That is the job I built WyrmSight to do.

6. Future considerations

These are the directions I am working toward. None of them is in the product as of the initial launch, and I am not promising any dates yet, but they are the directions that seem like a natural fit for the dataset WyrmSight already collects.

1. **Deeper data collection, tagging, and reporting.** I want WyrmSight to capture the pages behind the front door more deeply (e.g. a payment flow or an admin area) and tag each one so you can report on it separately. It builds on the scripted sessions that ship today. A payment page is one obvious use but my plan is to make this flexible so that it can work for a variety of situations that GRC folks will find useful.

2. **APIs, webhooks, and/or MCP.** I want it to be easy to get the data out and into the systems a team already runs, with an API to pull the data and perhaps webhooks to push alerts. The same API would let you ask questions of your data from an AI chat.

3. **A CSP workbench.** I want a place where you can build a new policy, or improve the one you have, using what WyrmSight has observed. It would explain a policy directive by directive, point out where it is weak, and suggest tighter settings based on what it saw (e.g. if no page ever loaded a frame, frames can be switched off). Any suggestion would cover only the pages WyrmSight has seen, which is one more reason for the first item on this list. I would also like it to rate the policy with the CSP Index, my public proposal for rating a CSP header from 0 to 10. The spec is public and open for argument (https://github.com/JGillam/csp-index), and a calculator that implements it runs in your browser at https://mechanicaldrake.com/lab/csp-index. That rating would apply to the policy text alone and would never feed a risk score for the application.

4. **Secret loss detection.** Today WyrmSight reports where data goes. I want it to recognize *what* is being sent. You would mark what counts as a secret in your application (i.e. anything potentially sensitive or not public), and WyrmSight would watch for those values reaching a third party. This is the oldest goal behind the product. In 2015 I published Paramalyzer, a Burp extension for parameter analysis on large penetration tests. It pulls parameters apart (decoding nested encodings along the way), helps identify sensitive data and hash types, and reports when a value marked secret shows up outside the application's scope (https://github.com/JGillam/burp-paramalyzer). I would like WyrmSight to do that same job without a pen tester at the keyboard.

About the author

I'm Jason Gillam. I have been in technology for nearly 30 years and in application security more than half of that. A lot of that time was spent on web application penetration testing and software development. Mechanical Drake is my company, focused on application security engineering.

WyrmSight is open at wyrmsight.com. There is a Free plan, and a two-week trial of the Professional plan when you sign up. If this paper describes a problem you live with, or you think I have something wrong, I would like to hear from you: hello@mechanicaldrake.com.

Appendix

Terms

- **First party.** The site's own domain. In WyrmSight, any domain the company has claimed as its own.
- **Third party.** Any domain a page contacts that isn't first party (e.g. an analytics vendor or an ad network).
- **Fourth party.** A third party the page never asked for directly. Another third party loaded it. WyrmSight lists these as third party, names the vendor that loaded them, and shows the chain. Its Depth column counts hops from the page, including hops through your own code, so depth alone does not make something a fourth party.
- **Tag manager.** A script that loads other scripts (tags) configured by whoever manages it, usually marketing, without a code change to the site.
- **Connection.** A domain an application talks to while a page loads.
- **Content Security Policy (CSP).** A response header that tells the browser which sources a page may load scripts, frames, connections and other resources from. In report-only mode the browser reports violations without blocking anything.

About the numbers

Every number in this paper comes from my own daily scans of 200 public household-name websites, which I picked by hand, weighted toward US and European consumer brands. The window is October 20 to 26, 2026: one cold load per site per day, landing page only, desktop browser from a US data center, no login, and nobody clicking the consent banner. 197 of the 200 sites gave usable captures. Each number is an aggregate across those sites.

A few things worth knowing when you read them:

- A single cold visit to a landing page is a floor on what a real visit does.
- "Third party" means a registrable domain other than the site's own. A brand's own CDN on a different domain counts as third party, so domain counts run a little high.
- Not every request records what loaded it, so the chain numbers (tag managers, fourth parties) are given as floors or ranges.

- The library number covers only versions a page exposes in a readable form. Advisories come from the public OSV database.

Capture quality is something I keep working on, and the sites themselves keep changing, so these numbers replace ones I have published before.

Sources

- PCI Security Standards Council, *Payment Card Industry Data Security Standard: Requirements and Testing Procedures*, v4.0.1, June 2024. Requirements 6.4.3 (p. 153) and 11.6.1 (p. 288).
- GDPR Article 30 (records of processing activities).
- CSP Index: spec at https://github.com/JGillam/csp-index; calculator at https://mechanicaldrake.com/lab/csp-index
- Paramalyzer: https://github.com/JGillam/burp-paramalyzer
- OSV, open source vulnerability database: https://osv.dev