Attack the Glass: Executive Summary

Executive Summary on the Attack the Glass (ATG) Vulnerability

v2.4.7, September 14, 2026

TLP:CLEAR

Contact: Robert Beckett

Signal: ndiligence.99

Email: atg@nDiligence.com

You do not control your own front page

Your organization publishes to screens. Customers, citizens, clinicians, traders, and operators make decisions based on what those screens show them. Your organization is accountable for that content, and you believe you control it, but for the last decade you might not.

When someone opens your application, their device does not receive a finished page. It receives instructions, then builds the page at runtime on their device using dozens or hundreds of components fetched live from other companies: content-delivery networks, analytics, advertising, sign-in and payment providers, and increasingly AI services. Whoever controls any one of those components decides what your screen shows, what your users see, and even how your application functions, all delivered through your own brand. They can also choose who receives real or modified capabilities. Nothing looks wrong to the person reading it, and it never lands in your logs because your servers were never touched.

This is Attack the Glass, a type of supply chain injection vulnerability that targets the last step in almost every digital system: the moment a screen is assembled and shown to a person. The page you publish is not the page your reader sees, and neither of you can tell.

Every control you own passed

Your security program is not failing. Your team built it to defend the systems your organization runs, and this attack never touches them. It happens on your customer's device, in code fetched from another company's server, because your application told that device to trust that company.

Every meaningful control your organization owns operates on a resource **before** it runs. Code review, signing, dependency scanning, and supply-chain attestation all examine files at rest. Subresource Integrity compares a fetched file against an expected hash as it arrives, and a Content-Security-Policy names which companies a browser may fetch from at all. Zero Trust, as NIST's guidance sets it out in seven tenets, authenticates and authorizes access to a resource, and its own language is "before access is allowed." An allowlist approves the sender, not the package.

No control re-examines the code once it starts running, so the change lands after every one of those controls has already passed and reported success. Your engineers will often find Subresource Integrity and a strict Content-Security-Policy available in the runtime and switched off in practice, and neither one would reach this far even switched on.

Detection is the usual answer when prevention fails, and here detection has nothing to work with. There is no malicious file to scan, because the code itself is often entirely unmodified. There is no fraudulent domain to block, because the delivery source is one your organization deliberately trusts. There is no anomalous log entry, because nothing anomalous happened on any system you operate. In the most advanced form of the attack, a fully verified, hash-valid script fetches its instructions while it is running, from data the attacker controls, so the script passes every check and the weapon is in the data rather than the code.

The user does not choose whom to trust

Your developers add a line of code that loads a script from another company. That company's script loads scripts from further companies, and those load more again. Your customer's browser runs all of it with your application's full permissions, because your application instructed it to. Your customer was never asked, cannot see which companies are involved, and cannot refuse any one of them without abandoning your product. Their only choice is to accept every company in that chain, sight unseen, or to stop using your product.

Your customer carries that risk first. When one of those companies serves altered code, your customer is the person who acts on a false screen: sends the payment, takes the dose, repeats something untrue. They cannot tell which company caused it, and they hold no contract with that company. Your organization carries the risk second, and only if somebody works out what happened: a complaint, a chargeback, a regulator's letter, a story in the press.

Your suppliers do this to you as well. Every provider whose console your staff signs into loads its own third-party code, chosen by developers you do not employ. Your engineers act on what an identity console or a cloud dashboard shows them, on the same terms your customers act on what your application shows them, and with the same inability to check.

Ask your own team which companies your application loads code from while it runs. Most organizations cannot answer, because no document anywhere records it. The answer lives scattered across script tags, tag manager configurations, and whatever dependencies those bring in behind them. Nobody approved the full set, because nobody has ever seen the full set.

The position is bought, not breached

In 2024 a company called Funnull bought polyfill.io, a web address more than a hundred thousand websites relied on for a piece of shared code. Funnull then served altered code from the same domain, through the same publishing channel those websites had always used. Nobody broke in, and nothing your intrusion detection is built to catch would have fired. The US Treasury sanctioned Funnull in 2025, finding that the company had bought a repository of code used by web developers and altered it to send visitors of legitimate websites to scam websites. The URL never changed. The person behind it did.

A precision operation once needed a state intelligence service behind it, because a human analyst had to research every target by hand. Generative AI does that research now, so content shaped for one recipient, or for one segment, costs whatever the compute costs. Criminal groups, commercial rivals, insiders, and state services all reach the same capability at the same price.

Who receives the change

An attacker weighs two things when choosing a target, and neither one predicts the other: how many people will receive the altered version, and how much authority sits behind the screen that receives it.

An attacker who controls one advertising or analytics component on your site never has to change the page for everybody. Within the same hour, from the same web address your readers have always used, that attacker sends the real page to most of your readers, an altered paragraph to readers whose connection places them in one country, a different alteration to readers whose browsing marks them as likely voters or likely buyers, and the real page again to any request that looks like a security scanner, a fact-checker, or a rival newsroom. Each reader sees one version and cannot see anyone else's. Your own servers handle none of it.

Your marketing team buys audiences this way every week. An advertising platform already sells targeting by country, device, time of day, past purchases, and membership of a commercial or political segment, and those same controls decide who receives the altered page. Reaching a hundred thousand people who share one characteristic costs an attacker no more effort than reaching one person, because advertising companies built the infrastructure to do exactly that.

A head of government reads from one screen. A bank's chief executive reads from another. A commander directing a force reads from a third. An attacker who reaches only one of those screens has picked the smallest audience the mechanism allows and, in the same

move, the largest consequence it can produce, because the authority of the person reading sets the ceiling. Targeting one named individual on one device is the **precision extreme**, the finest aim available, and a decision any of those three takes from a false picture needs no second recipient to matter.

An attacker picks how many people to reach and whose screen to reach as two separate choices, and neither one limits the other. Narrowing the audience to one narrows nothing except the number of people who could have noticed.

Five effects: deceive, disrupt, degrade, deny, destroy

An attacker holding one position in your supply chain can produce five distinct effects on your screen. Each one works on its own, and nothing stops an attacker combining them.

Deceive. A trader sees a position that is not the position, or a clinician sees a lab value that nobody reported. The person is authorized and the action is authorized, but the information is false.

Disrupt. A feature still looks like it works and quietly does nothing. Your password reset completes and sends no email. Your fraud report button returns a confirmation and files no report.

Degrade. Everything still works, and works worse. Your checkout slows until customers abandon it, and your team books the loss as a conversion problem rather than a security one.

Deny. The page, the route, or the control is not reachable at all, which is the failure your customers telephone you about.

Destroy. Code reaches through the screen into a database, a controller, or a storage account behind it, and damages what it finds.

The systems behind your screen set the ceiling, not the attacker's skill.

What sits behind your screen

A hospital, a trading floor, a newsroom, and a substation all assemble their screens the same way, and they differ only in what sits behind the screen and who acts on what it shows. We reason the sections below from how these systems are built, because no public incident yet documents each sector separately.

News, media, and publishing. Your masthead is the asset, and an attacker reaches it directly. Readers can be shown an altered version of your reporting carrying your byline and

your credibility, while your editors, your fact-checkers, and any archive you consult all see the correct version. You cannot retrieve what any individual reader was shown, because no copy of it exists anywhere you control.

Financial services. Financial services carry two exposures, and a regulator has already recorded the first. Attackers have skimmed payment and checkout pages this way for a decade, and in the Ticketmaster case the attacker never touched Ticketmaster at all, but instead compromised a third party whose chat script sat on the payment page, which a UK regulator recorded in a penalty notice. The second exposure is internal, because a treasury or trading officer acting on a false position summary is an authorized user taking an authorized action on information they had every reason to trust.

Healthcare. Clinical decision support, medication administration, results review, and device interfaces are all rendered surfaces. A clinician reading a value has nowhere else to check it against, and often no clinician reads it at all: an alerting rule reads it instead, and does not fire.

Energy, utilities, and industrial operations. Elsewhere an attacker changes what a person believes. Here an attacker changes what a machine does. A page that exists to be read gives an attacker little to break. An operator's control screen commands pumps, breakers, valves, interlocks, and alarms, so breaking what that screen does matters more than changing what it says. Where a system must answer within a fixed time, a delay that would merely irritate a web user makes a control action arrive too late to work. Well-run plants narrow what reaches those screens through pinning, network segmentation, and hardened software, but none of that reaches back to the companies the components came from.

Government and the public sector. Benefits portals, tax systems, public health guidance, and emergency information are the channel through which citizens receive the state's word. Officials, analysts, and administrators also make decisions from screens assembled the same way, so the exposure runs in both directions.

Every large organization, whatever the sector. Your own administrators sit in front of the same problem, and most executives have never considered them. The signed-in account's own permissions set the ceiling, not the attacker's skill, so the same substitution that reaches a customer's belief on a checkout page also reaches the directory on an identity console, the tables on a database administration interface, the storage on a cloud console, and the build on a CI/CD interface. Those consoles are client-side rendered applications, assembled at runtime from the same commercial supply chain as your marketing site.

What bounds it

An attacker gains no new permissions from any of this. Your trading application can already move money, so an attacker who changes the position it displays gets money moved by an authorized trader who read a false number. Your public marketing page can do almost nothing, so an attacker who changes it misleads a reader and goes no further.

Whatever the signed-in person is already allowed to do is the most an attacker can achieve through their screen. A read-only page lets an attacker deceive a reader and nothing more. A database administrator's console lets an attacker deceive, disrupt, degrade, deny, and destroy.

An administrator's console is also the screen fewest people use and the hardest one to reach. An attacker who goes after it takes on a smaller, better-defended target in exchange for a larger result.

What we know, and what nobody can see

We separate what we can prove from what we infer, what we assume, and what nobody can measure.

Established. The mechanism is real, and the incidents that have used it are documented in the public record.

Reasoned. Severity scales with what sits behind the screen, which follows from how these systems are built rather than from a documented case in every sector.

Presumed. Capable actors already sit in the positions an operation would need. Our threat-intelligence work places actors where such an operation would run, but those findings sit in restricted channels you cannot check, which is why we mark the conclusion presumed and why no other claim in this document rests on it.

Unknown. Nobody can establish the true scale of operational use today, because the instrumentation that would measure it does not exist.

We cannot tell you an operation is running against you, and no tool in standard use can tell you one is not. Absence of evidence is not evidence of absence, and here the absence is a property of the attack rather than a sign of safety. That is not a hedge. It is the finding, and it is why awareness is the mitigation available now.

Why you are hearing this now

We briefed governments before we briefed anyone else. US government briefings began in April 2024, before any public disclosure, and continued through 2026 to national and multinational cyber authorities. In August 2026 we sent formal written notice of intent to publish to every briefed body, thirty-five days ahead of release, with a final offer to coordinate.

No detection capability emerged in that time. A real capability, no way to detect its use, and capable actors already in position: that combination is why we publish on September 14 instead of keeping this in restricted channels. Awareness is the defense that exists today, and awareness only works in the open.

Five questions worth asking your team

1. Which third parties do our applications fetch executable code from at runtime, and who do *those* parties fetch from?
2. If one of our providers changed ownership tomorrow, how would we find out, and how long would that take?
3. Do our administrative and operator consoles load third-party code at runtime, and does anyone review what they load?
4. If a customer told us they saw something on our site that we never published, could we prove or disprove it?
5. Which of our controls would detect a change that arrives from an approved source and executes after every check has passed?

If nobody in your organization can produce a number for the first question, you do not need to ask the other four.

About the Work

nDiligence is a diligence, security, and threat intelligence firm, formed in March 2026 and built on investigative and security research underway since 2020. Attack the Glass is original nDiligence research, self-funded, first identified in 2022, and drawn entirely from publicly available information. A provenance and methodology appendix maps every material claim to its public source, files each against one of four registers of certainty, and states plainly what could not be established. It makes no original attribution: the operations it cites are publicly documented, and any actor it names is named as the public record already names them. The full technical documentation, primers, and that appendix are at **https://atg.nDiligence.com**.

Contact:

Robert Beckett

Founder and CEO, nDiligence

Email: atg@nDiligence.com

Signal: ndiligence.99