Logjam: the latest TLS vulnerability explained

2015-05-20

Filippo Valsorda

6 min read

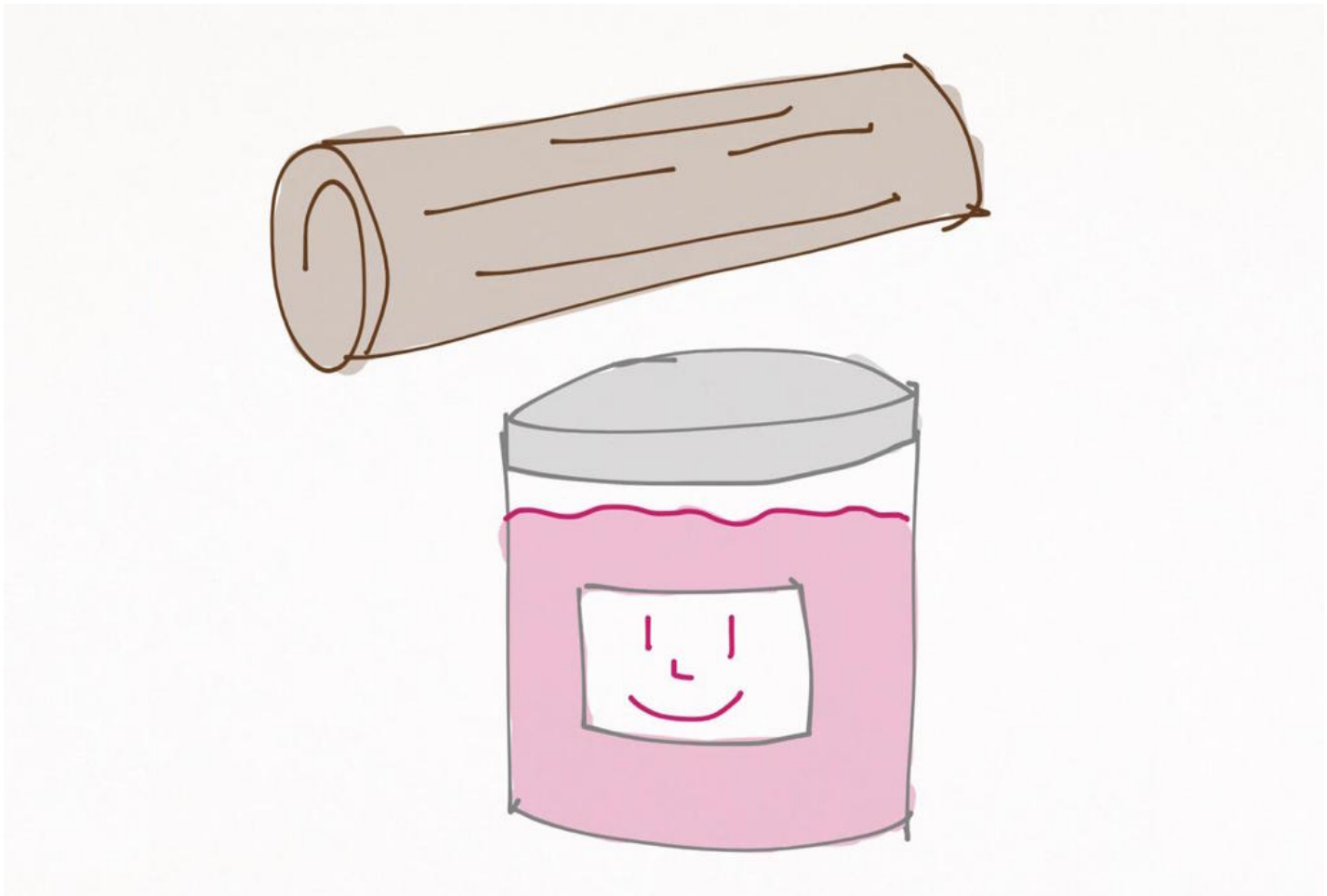


Image: "Logjam" [as interpreted by @0xabad1dea]

Yesterday, a group from INRIA, Microsoft Research, Johns Hopkins, the University of Michigan, and the University of Pennsylvania published a deep analysis of the Diffie-Hellman algorithm as used in TLS and other protocols.

This analysis included a novel downgrade attack against the TLS protocol itself called **Logjam**, which exploits EXPORT cryptography (just like FREAK).

First, let me start by saying that **CloudFlare customers are not and were never affected**. We don't support non-EC Diffie-Hellman ciphersuites on either the client or origin side. We also won't touch EXPORT-grade cryptography with a 20ft stick.

But why are CloudFlare customers safe, and how does Logjam work anyway?

Diffie-Hellman and TLS

This is a detailed technical introduction to how DH works and how it's used in TLS—if you already know this and want to read about the attack, skip to "Enter export crypto, enter Logjam" below. If, instead, you are not interested in the nuts and bolts and want to know who's at risk, skip to "So, what's affected?"

To start a TLS connection, the two sides—client (the browser) and server (CloudFlare)—need to agree securely on a secret key. This process is called **Key Exchange** and it happens during the TLS Handshake: the exchange of messages that take place before encrypted data can be transmitted.

There is a detailed description of the TLS handshake in the first part of this previous blog post by Nick Sullivan. In the following, I'll only discuss the ideas you'll need to understand the attack at hand.

There are many types of Key Exchanges: static RSA, Diffie-Hellman (DHE cipher suites), Elliptic Curve Diffie-Hellman (ECDHE cipher suites), and some less used methods.

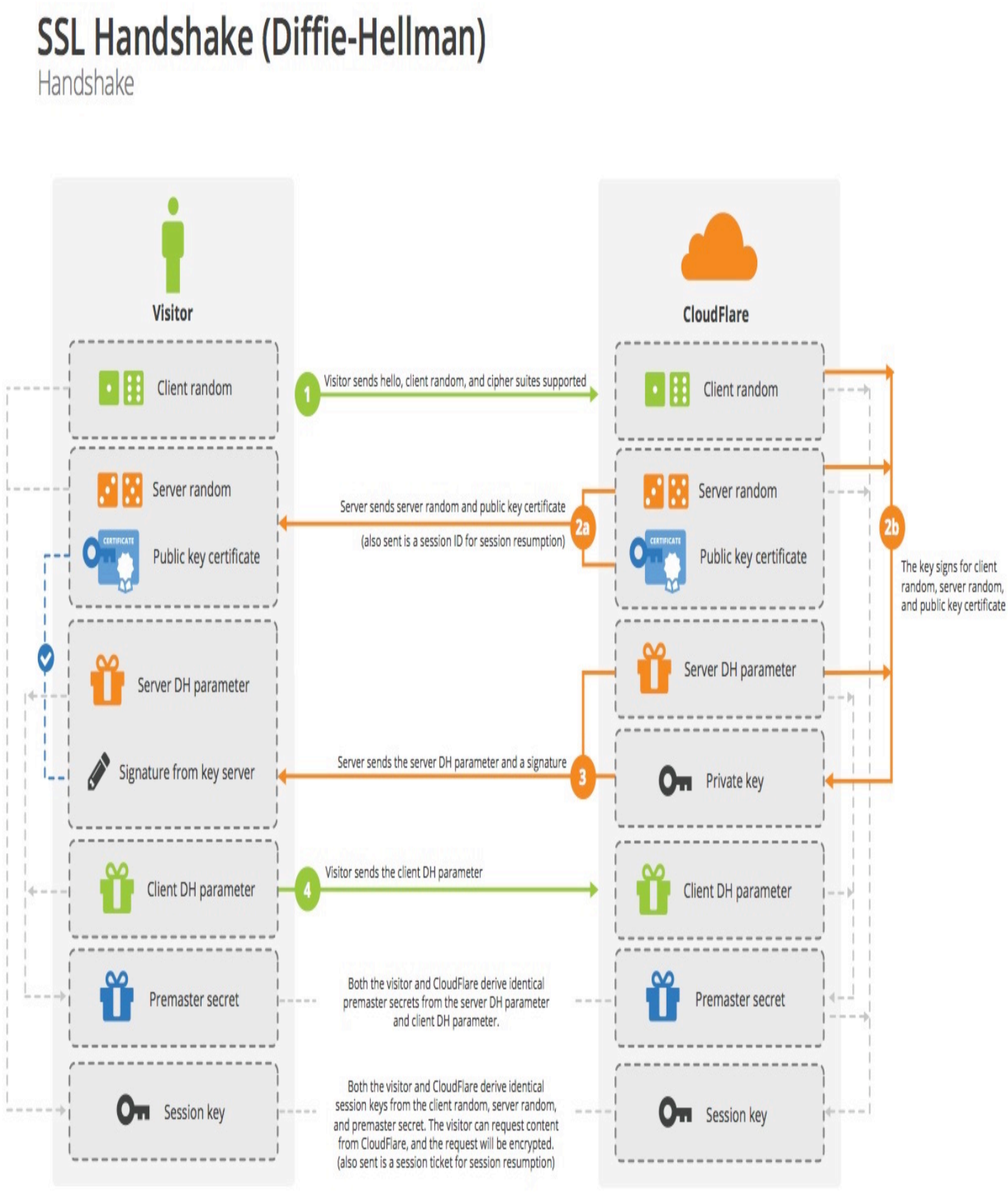
An important property of DHE and ECDHE key exchanges is that they provide Forward Secrecy. That is, even if the server key is compromised at some point, it can't be used to decrypt past connections. It's important to protect the

information exchanged from future breakthroughs, and we're proud to say that 94% of CloudFlare connections provide it.

This research—and this attack—applies to the normal non-EC **Diffie-Hellman key exchange (DHE)**. This is how it works at a high level (don't worry, I'll explain each part in more detail below):

1. The client advertises support for DHE cipher suites when opening a connection (in what is called a Client Hello message)
2. The server picks the parameters and performs its half of the DH computation using those parameters
3. The server signs parameters and its DH share with its long-term certificate and sends the whole thing to the client
4. The client checks the signature, uses the parameters to perform its half of the computation and sends the result to the server
5. Both parts put everything together and derive a shared secret, which they will use as the key to secure the connection

(For a more in depth analysis of each step see the link to Nick Sullivan's blog post above, "Ephemeral Diffie-Hellman Handshake" section.)



Let’s explain some of the terms that just passed by your screen. “The client” is the browser and “the server” is the website (or CloudFlare’s edge serving the website).

"The parameters" (or *group*) are some big numbers that are used as base for the DH computations. **They can be, and often are, fixed. The security of the final secret depends on the size of these parameters.** This research deemed 512 and 768 bits to be weak, 1024 bits to be breakable by really powerful attackers like governments, and 2048 bits to be a safe size.

The certificate contains a public key and is what you (or CloudFlare for you) get issued from a CA for your website. The client makes sure it's issued by a CA it trusts and that it's valid for the visited website. The server uses the corresponding private key to cryptographically sign its share of the DH key exchange so that the client can be sure it's agreeing on a connection key with the real owner of the website, not a MitM.

Finally, the DH computation: there's a beautiful explanation of this on Wikipedia which uses *paint*. The tl;dr is:

1. The server picks a secret 'a'
2. Then it computes—using some parameters as a base—a value 'A' from it and sends that to the client (not 'a'!)
3. The client picks a secret 'b', takes the parameters from the server and likewise it computes a value 'B' that sends to the server
4. Both parts put together 'a' + 'B' or 'b' + 'A' to derive a shared, identical secret - which is impossible to compute from 'A' + 'B' which are the only things that travelled on the wire

The security of all this depends on the strength/size of the parameters.

Enter export crypto, enter Logjam

So far, so good. Diffie-Hellman is nice, it provides Forward Secrecy, it's secure if the parameters are big enough, and the parameters are picked and signed by

the server. So what's the problem?

Enter "export cryptography"! **Export cryptography** is a relic of the 90's US restrictions on cryptography export. In order to support SSL in countries to where the U.S. had disallowed exporting "strong cryptography", many implementations support weakened modes called EXPORT modes.

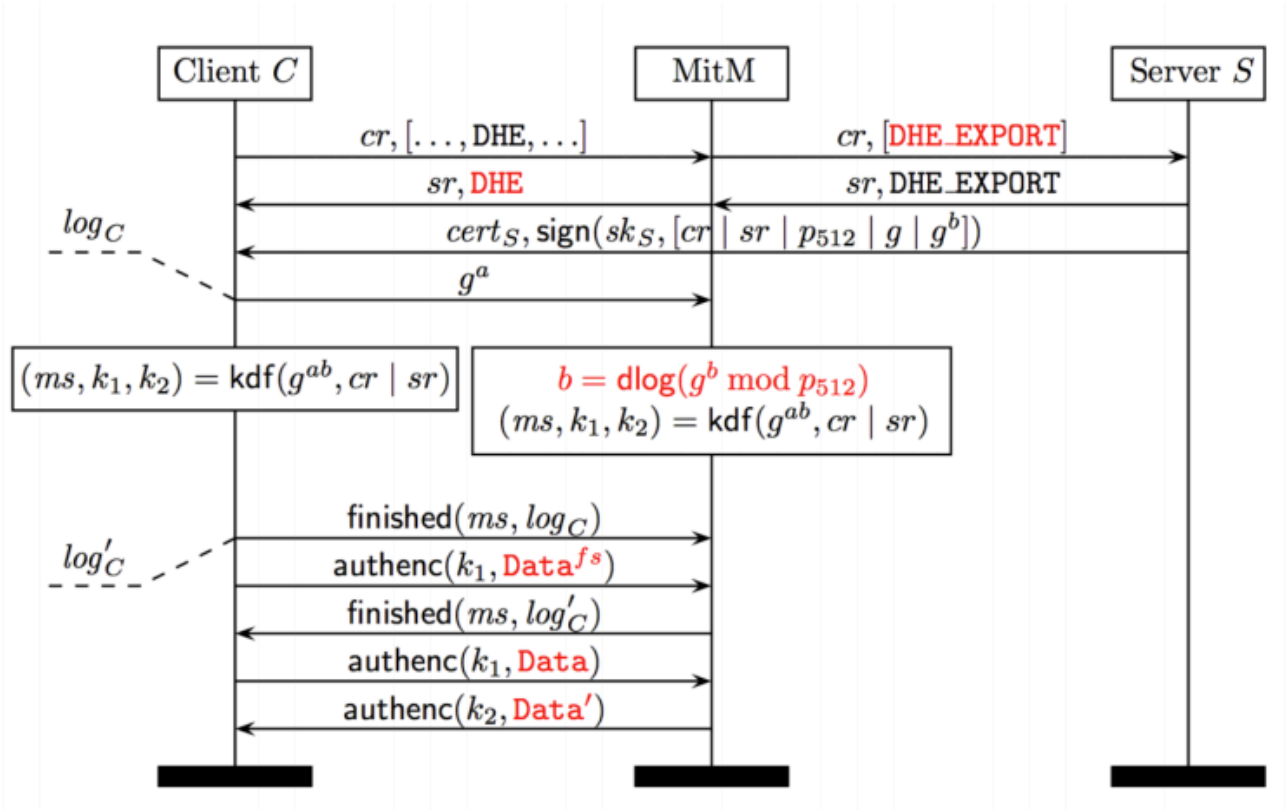
We've already seen an attack that succeeded because connections could be forced to use these modes even if they wouldn't want to, this is what happened with the FREAK vulnerability. It's telling that 20 years after these modes became useless we are still dealing with the outcome of the added complexity.

How it works with Diffie-Hellman is that the client requests a `DHE_EXPORT` ciphersuite instead of the corresponding `DHE` one. Seeing that, the server (if it supports DHE_EXPORT) *picks small, breakable 512-bits parameters* for the exchange, and carries on with a regular DHE key exchange. **The server doesn't signal back securely to the client that it picked such small DH parameters because of the EXPORT ciphersuite**.

This is the protocol flaw at the heart of Logjam "downgrade attack":

- A MitM attacker intercepts a client connection and replaces all the accepted ciphersuites with only the DHE_EXPORT ones
- The server picks weak 512-bits parameters, does its half of the computation, and signs the parameters with the certificate's private key. **Neither the Client Hello, the client ciphersuites, nor the chosen ciphersuite are signed by the server!**
- The client is led to believe that the server picked a DHE Key Exchange and just willingly decided for small parameters. From its point of view, it has have no way to know that the server was tricked by the MitM into doing so!

- The attacker would then break one of the two weak DH shares, recover the connection key, and proceed with the TLS connection with the client



Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice - Figure 2

The client has no other way to protect itself besides drawing a line in the sand about how weak the DHE parameters can be (e.g. at least 1024 bits) and refuse to connect to servers that want to pick smaller ones. This is what all modern browsers are now doing, but it wasn't done before because it causes breakage, and it was believed that there was no way to trick a server into choosing such weak parameters if it wouldn't normally.

The servers can protect themselves by refusing EXPORT ciphersuites and never signing small parameters.

About parameters size and reuse

But how small is too small for DH parameters? The Logjam paper analyzes this in depth also. The first thing to understand is that **parameters can be and often are reused**. 17.9% of the Top 1 Million Alexa Domains used the same 1024-bit parameters.

An attacker can perform the bulk of the computation having only the parameters, and then break any DH exchange that uses them in minutes. So when many sites (or VPN servers, etc.) share the same parameters, the investment of time needed to "break" the parameters makes much more sense since it would then allow the attacker to break many connections with little extra effort.

The research team performed the precomputation on the most common 512-bit (EXPORT) parameters to demonstrate the impact of Logjam, but they express concerns that real, more powerful attackers might do the same with the common normal-DHE 1024-bit parameters.

Finally, in their Internet-wide scan they discovered that many servers will **provide vulnerable 512-bit parameters even for non-EXPORT DHE**, in order to support older TLS implementations (for example, old Java versions).

So, what's affected?

A client/browser is affected if it accepts small DHE parameters as part of any connection, since it has no way to know that it's being tricked into a weak EXPORT-level connection. Most major browsers at the time of this writing are vulnerable but are moving to restrict the size of DH parameters to 1024 bit. You can check yours visiting weakdh.org.

A server/website is vulnerable if it supports the DHE_EXPORT ciphersuites or if it uses small parameters for DHE. You can find a test and instructions on how to fix this at https://weakdh.org/sysadmin.html. 8.4% of Alexa Top Million HTTPS websites were initially vulnerable (with 82% and 10%

of them using the same two parameters sets, making precomputation more viable). CloudFlare servers don't accept either DHE_EXPORT or DHE. We offer ECDHE instead.

Some interesting related statistics: **94% of the TLS connections to CloudFlare customer sites uses ECDHE** (more precisely 90% of them being `ECDHE-RSA-AES` of some sort and 10% `ECDHE-RSA-CHACHA20-POLY1305`) and provides Forward Secrecy. The rest use static RSA (5.5% with AES, 0.6% with 3DES).

Both the client and the server need to be vulnerable in order for the attack to succeed because the server must accept to sign small DHE_EXPORT parameters, and the client must accept them as valid DHE parameters.

A closing note: **events like this are ultimately a good thing for the security industry and the web at large** since they mean that skilled people are looking at what we rely on to secure our connections and fix its flaws. They also put a spotlight on how the added complexity of supporting reduced-strength crypto and older devices endangers and adds difficulty to all of our security efforts.

If you've read to here, found it interesting, and would like to work on things like this, remember that we're hiring in London and San Francisco!

Y **Discuss on Hacker News**

TLS Vulnerabilities Security

Follow on X

Filippo Valsorda | @filosottile

Cloudflare | @cloudflare

RELATED POSTS

June 24, 2026

Unlocking the Cloudflare app ecosystem with OAuth for all

Self-Managed OAuth is now available to all developers on Cloudflare. Here's how we executed a zero-downtime migration of our core OAuth engine to make it happen....

By Sam Cabell, Mike Escalante, Adam Bouhmad, Nick Comer

Developers, API, Security, OAuth, Developer Platform, Agents, Product News, Cloudflare Media Platform, Identity

June 23, 2026

The White House's post-quantum executive order is an important milestone. It’s time to get to work

The new executive order sets a 2030 migration deadline and establishes a powerful foundation for post-quantum resilience. We look at what it gets right, where it can go further, and our migration playbook for government and industry....

By Sharon Goldberg, Vincent Voci

Post-Quantum, Security, Cryptography, Policy & Legal, Government Innovation, Impact

June 18, 2026

Build your own vulnerability harness

We break down the technical architecture behind our multi-stage vulnerability discovery harness and automated triage loop. Learn how we manage state controls, squash false positives through adversarial review, and route around LLM context limits....

By Dan Jones, Alexandra Godoi, Grant Bourzikas

Security, AI, Engineering, Research, Vulnerabilities

June 09, 2026

Defend against frontier cyber models: Cloudflare's architecture as customer zero

In our post about Project Glasswing, we made the argument that the architecture around a vulnerability matters more than the speed of the patch. Here we walk through what that architecture looks like, the threats it defends against, and how we run it ourselves as Cloudflare's customer zero....

By Rohit Chenna Reddy, Chase Catelli, Dan Jones

Security, AI, Threat Intelligence, Risk Management, Customer Zero, WAF, Zero Trust, Cloudforce One, Bot Management