
Opener of The Ways: Protecting Against Malicious Website Crawlers Via DID Key-Based Authentication

Michael Mulet Miles Mulet
Independent Researcher Independent Researcher

With
Apart Research

Abstract

Opener of the ways (ootws) is a distributed invite-only identity framework to protect open source infrastructure, like blogs, git forges, and personal websites from attacks from AI swarms, and over eager web crawlers. Using distributed-identities ootws only serves expensive resources to trusted users. It also provides a way to do distributed, trusted banning. If a user disobeys resource limits or makes too many requests, it can provide verified proof of misbehavior, allowing other servers to ban that malicious user.

Keywords: Distributed-id, AI swarm protection, AI web crawler protection, open source protection

1. Introduction

Modern AI systems are trained using a large amount of data, most of which is scraped from internet websites. Recently, a certain class of malicious crawlers have infamously ignored website rules (such as robots.txt) and perform an unreasonable amount of requests to servers, **increasing the costs** of hosting a website ([Martinus](#)).

While solutions like Cloudflare exist, **open-source is the way man**. Many in the open source community are skeptical of relying entirely on one company. There needs to exist a system which disallows unwanted clients from accessing data, while allowing access to rule-following entities.

2. Related Work

To combat malicious clients, several different methods have been proposed.

[Anubis](#) and similar techniques issues proof-of-work challenges to clients in an attempt to raise the cost of scraping a website. Unfortunately, powerful malicious clients have already demonstrated ways to circumvent this preventative measure ([Codeberg](#)). And maintaining anubis puts undo stress on the creator <https://news.ycombinator.com/item?id=44915499> (“I haven't been able to put as much energy into Anubis as I've wanted because I've been incredibly overwhelmed by life and need to be able to afford to make this my full time job. Support contracts are being roadblocked, and I just wish I had the time and energy to focus on this without having to worry about being the single income for the household.”)

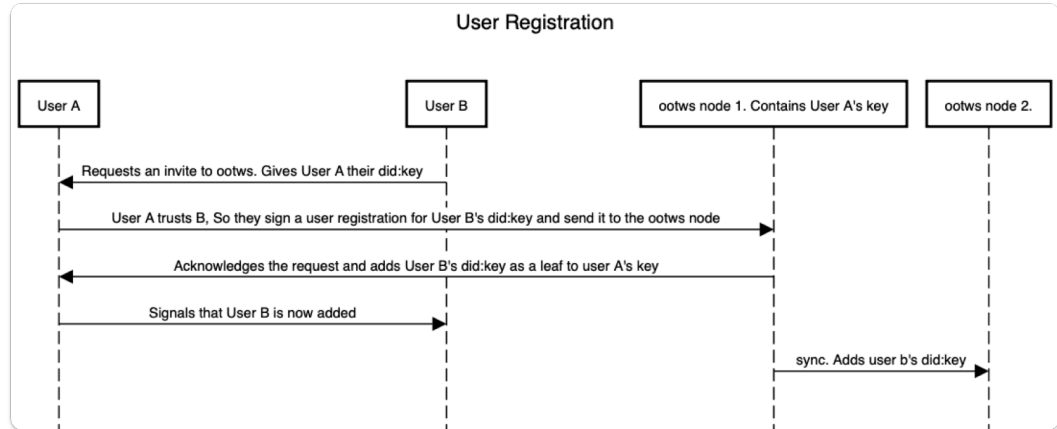
[Nepenthes](#) and similar techniques likewise attempt to raise the cost of malicious clients by serving endless nonsense pages in locations that would normally be inaccessible to legitimate clients. However, it is noted that sophisticated bots already discovered how to escape such “tarpits” ([Belanger](#)), and in the future it is widely expected that malicious actors' capabilities will grow as AI technology continues to develop and become widely distributed.

What these methods have in common is that they both exemplify a “cat and mouse game”, where malicious clients may adapt to and learn from failed attempts, necessitating more sophisticated methods from defenders.

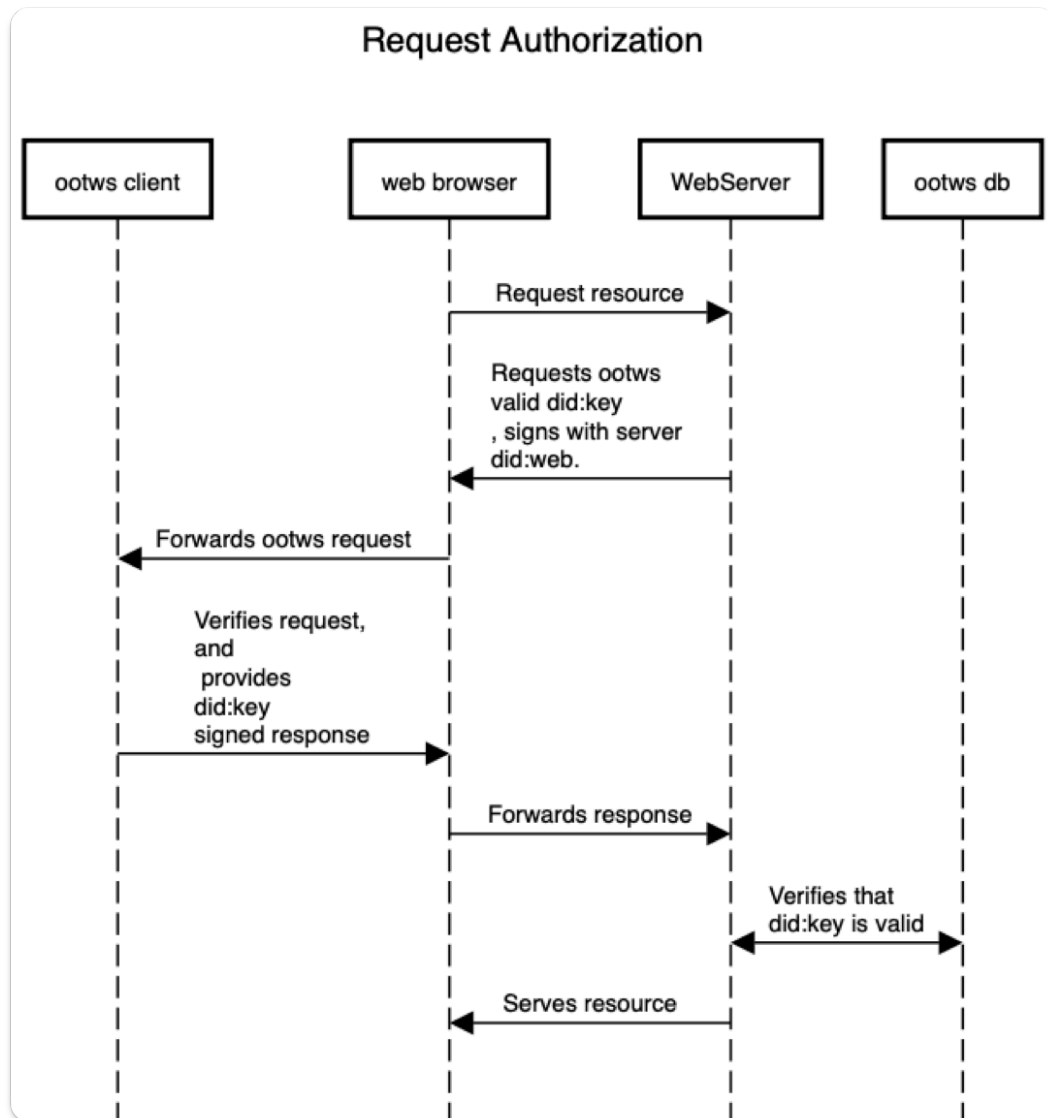
This won't scale as AI capability increases. Intelligence at scale will defeat all “cat-and-mouse”, so we need a cryptographically secure way to defend open source.

3. Methods

We present a protocol to allow for trusted clients to access websites while preventing malicious clients from making unreasonable demands. Our system is invite-only based and distributed, based on DID keys ([Longley](#) et al). DID keys all use public-private key infrastructure so they can sign arbitrary information, so we can be sure of its origin. Our system is distributed into a group of servers, each server is called a “node”. Each node has a complete list of trusted DID keys in an easily shareable sqlite database. Each user who has a key in the database can invite other users to add their key. This is kept track of in the database as a tree structure. If a user acts maliciously (goes where it shouldn't, goes beyond rate limits, or bandwidth limits) the node can then revoke the access to the user, and potentially the user who invited that user. Using DID's keep the system completely anonymous and free of censorship (no way to prove who is doing what).



Protected Resource request flow:



The user (on website or client), make a request to a server which has an active copy of the sqlite database. The server (using its did key) signs a ResourceRequest

which contains the url of the resource, and site-wide limits on bandwidth and request limits. It then sends the request back to the website. The website then forwards the request to a local client which verifies that the signature is correct, and it decides if accessing the resource would push it over the resource limits. If it is over the limit, it will refuse the request and processing stops there. If it is under the limit, it will sign an AuthorizationRequest that includes the original signed ResourceRequest and the DID. It sends this to the website which then forwards it to the server. The server then looks at the AuthorizationRequest and verifies that it has been properly signed and that the user who signed it is in good standing (it is in the sqlite database, and not banned and its parent is not banned). Then, and only then will it serve the requested resource.

Sync Flow

Because of the signed AuthorizationRequests and ResourceRequests, we have an audit trail of resource load usage per user. All cryptographically signed. The server can now sync to other nodes with their databases and provide proof of a malicious user to be revoked from their chains.

4. Results

We present the results of a synthesized trial-run [here](#).

Figure 1 – Representation of benchmarking Number Comprehension Conflation

5. Discussion and Conclusion

A good question might be, how do we bootstrap this invite only system. Won't it be hard to grow? The answer is: not at all! Such a system already exists in <https://lobste.rs/> an invite only web forum for software developers and open source enthusiasts with over 20,000 members. It is well known and popular among influential open source developers like Xe (the maintainer/creator of Anubis) and simonw. We can seed the invite only system, by distributing keys to the existing members of lobste.rs. Ootws itself can be thought of as a distributed version of the lobste.rs invite system.

I'm running out of time to write this report, but if you look at the tests you will see that this system also works if the website and/or node server are malicious and want to purposely get a user banned. The user client signing prevents them from doing so.

By using this system, backstabbing actors are publicly “shamed”, while good boys and girls may access content. This does not preclude the use of AI web-crawlers; it only precludes the ones a given distributed society deems bad boys and girls. In this

way, quality information may be shared where a community agrees together. That's usually good; not agreeing is bad.

6. References

Xe (2025) *Anubis*, <https://github.com/TecharoHQ/anubis>
Dave Longley, Dmitri Zagidulin, Manu Sporny (2025) *The did:key Method v0.9*, <https://w3c-ccg.github.io/did-key-spec/>
Codeberg, *It seems like the AI crawlers learned how to solve the Anubis challenges*. <https://social.anoxinon.de/@Codeberg/115033790447125787>
Herman Martinus. *The Great Scrape*
<https://herman.bearblog.dev/the-great-scrape/>
Prince, Matthew (2025) Cloudflare outage on November 18, 2025
<https://blog.cloudflare.com/18-november-2025-outage/>
Longley <https://w3c-ccg.github.io/did-key-spec/>

7. Appendix

Important: Include an appendix called "Security Considerations" that outlines potential limitations of your approach and suggestions for future improvements.

Security Considerations

Like any reputation-based system, the system may be exploited one-time by a previously trusted actor who **backstabs you**.

However, our system **publicly shames and hurts backstabbers**.