Matching IOCs is now fast and easy!

Seth Hall

Zeek Workshop at CERN • Geneva • March 2026

Seth Hall

Zeek Core Architect

Long-time contributor to Zeek's architecture and development

Corelight Co-Founder

Built the company around Zeek for enterprise network security

MatchyLabs

Building high-performance security data tools (and other stuff!)

MatchyLabs

What I've been building

Matchy

High-performance IOC matching engine

Zeek Plugin

Drop-in replacement for the Intel Framework

WalterOps

AI-powered system operations (walterops.com)

Today's focus: Matchy + Zeek

The Intel Framework

Every Zeek worker loads its own copy of the data

Data updates require reloading across all workers

Large indicator sets strain memory on big clusters

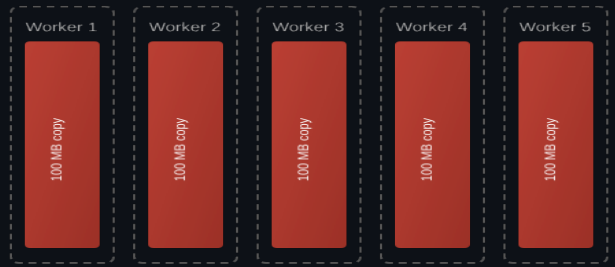
Complex cluster synchronization via Broker

These problems scale with your data and your cluster size

The Memory Problem

What happens on a real cluster

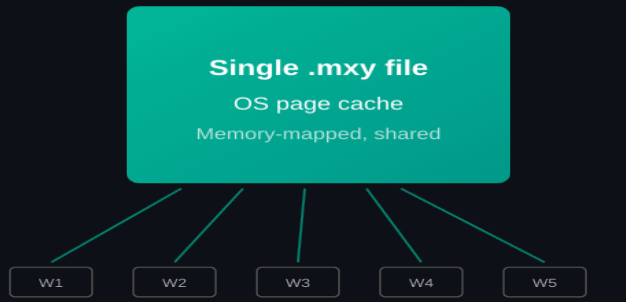
Intel Framework



... and so on for each worker

50 workers × 100 MB
= 5,000 MB RAM

Matchy (mmap)



50 workers → 100 MB
N workers, 1 copy

vs

OS shares physical memory pages across all processes via mmap

What if IOC matching could be different?


One database file, shared by every process


Load time under 1ms — regardless of size


IPs, strings, and wildcards in one query

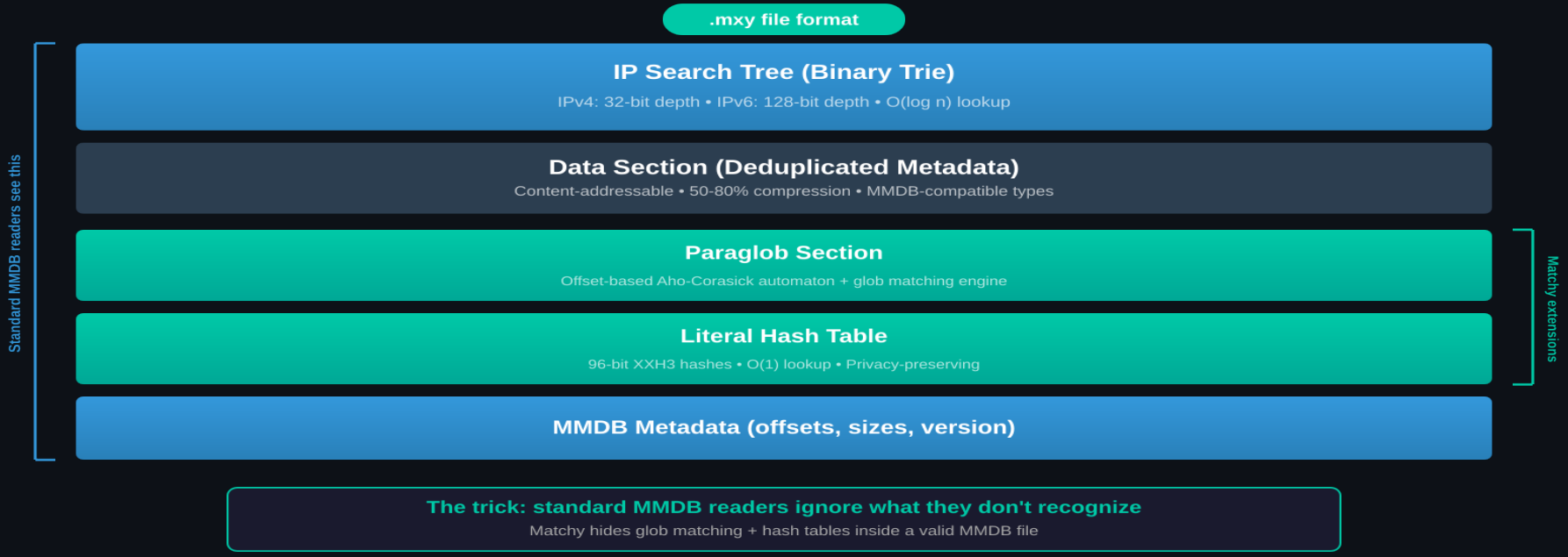

Hot-reload without restarting anything

This is Matchy.

Challenges

Database Structure

MaxMindDB format extension



Offset-Based Aho-Corasick

Making a classic algorithm mmap-friendly

The Challenge

- Standard Aho-Corasick uses pointers
- Pointers can't survive mmap — addresses change
- Rebuilding the automaton at load defeats the purpose

The Solution

- Replace all pointers with file offsets (u32)
- Automaton lives directly in the file
- mmap it → ready to query instantly

"Why is this database taking longer to load?"

The realization: some data was being copied to the heap at load time.

The fix: restructure everything so the database file IS the in-memory representation. No deserialization. No heap allocation. Just mmap and go.

Before	After
Load time grew with DB size	**< 1ms always. Period.**

"Do we actually need the original strings?"

The literal hash table stored every indicator string alongside its hash. That's a lot of data — and it's data we don't actually need to query.

With a 96-bit XXH3 hash, the probability of a false positive is ~10^{-24} per query. So we dropped the string pool entirely.

Before

`{hash, original_string, metadata}`

Strings dominated file size

After

`{hash_lo: u64, hash_hi: u32, id: u32}`

~50% smaller databases

Performance

< 1 µs	O(1)	450 MB/s	< 1 ms
IP lookup	**Exact string match**	**IOC extraction**	**Database load**
Binary trie traversal	96-bit XXH3 hash	SIMD-accelerated	Any size, always

23,000 lines of Rust • 10 specialized crates • 426 commits

Zeek!

Bringing it to Zeek: MatchyIntel

zeek-matchy-plugin • Full Intel Framework parity — and then some

Connections	Originator + responder IPs
DNS	Query domains
HTTP	URLs, Host, Referer, X-Forwarded-For, User-Agent
SSL/TLS	SNI, certificate CN
SMTP	Email addrs, header IPs, URLs in message bodies
x509 Certs	SAN domains, emails, cert hashes
Files	File hashes, filenames, SMB filenames
SSH	Server host key hashes

Plus: extensible via MatchyIntel::seen() for custom observations

Getting Started

Three steps to threat matching

1

Build your database

```
matchy build threats.csv -o threats.mxy    # or from MISP feeds!
```

2

Install the plugin

```
zkg install https://github.com/matchylabs/zeek-matchy-plugin
```

3

Add to local.zeek

```
@load Matchy/DB/intel
redef MatchyIntel::db_path = "/opt/threats.mxy";
```

Live Database Updates

Change your threat data without restarting Zeek

Update .mxy file

Build new database from your pipeline

→

Matchy detects change

File watcher triggers automatic reload

→

Hot swap

New data active lock-free, ~1-2ns overhead

→

Zero downtime

No restart needed no packets missed

```
# Atomic update: build to temp, mv into place
matchy build updated.csv -o threats.mxy.tmp && mv threats.mxy.tmp threats.mxy
```

Intel Framework vs Matchy

	Intel Framework	Matchy
Memory model	Per-worker heap copies	Shared mmap (N workers, 1 copy)
Load time	Grows with data size	< 1ms always
Data updates	Reload across workers	Hot-swap, zero downtime
Indicator types	Type-specific fields	Unified: IP + string + glob
Metadata	Limited fixed fields	Arbitrary JSON
Data sources	Zeek input framework	CSV, JSON, MISP feeds
Install	Built-in	zkg install (one command)

Matching IOCs is now fast and easy.

github.com/matchylabs/matchy

github.com/matchylabs/zeek-matchy-plugin

matchylabs.com

walterops.com

Seth Hall • seth@matchylabs.com • @sethhall