Blog.Dataparty

22. April 2023

WTF Is A KDF?

Earlier this week a letter from an activist imprisoned in France was posted to the internet. Contained within Ivan Alococo's dispatch from the Villepinte prison was a startling revelation. Police had cracked his LUKS hard drive password. A feat that once was impossible can now be accomplished in a few months by harnessing as many as 10,000 servers with modern GPUs. At the root of this breach is a cryptographic function that is showing its age, PBKDF2.

This episode is a wake up call to learn wtf is a KDF?

What Is A KDF?

In modern computing when applications provide strong file encryption they frequently use passwords to protect files. For passwords to be strong they must contain a lot of entropy and generally appear as random as possible. Obviously humans tend to use characters and phrases from their native language combined with memorable patterns or rules that help them remember these passwords.

Key derivation functions (KDFs) are tools that allow us to improve the entropy derived from the types of passwords people typically use. By performing a series of hashing and salting KDFs season the user's input with entropy sufficient for use in private keys for algorithms like AES and NaCl.

In the case of this French prisoner they were using Linux's most popular hard drive encryption tool, LUKS, which was using a PBKDF2 to generate AES keys. In Ubuntu 18.04 this is the default configuration. PBKDF2 is a password based KDF designed to be resistant to CPU based attacks and dates back to 2000. It was first mentioned as an internet standard in RFC-2898 in September 2000.

A Startling Revelation

Since the time PBKDF2 was designed, we've seen the rise of powerful GPUs become common place. To defend against this rising onslaught of GPU hashing powering is a relatively new algorithm, argon2.

argon2 sneaks up on pbkdf2

How Does Argon2 Work?

The cryptographic power of argon2 is sublte but brilliant. Instead of focusing on CPU time by requiring large numbers of hash iterations, argon2 wages war on your GPUs memory capacity. When hashing a password with argon2 an application developer can dial up the amount of RAM that is required to complete the computation. In so doing it starves the globs of highly parallel computation cores in a GPU reducing the total processing power the GPU can bring to bear.

Why does this work? On modern GPUs the exact number of threads that are active in parallel varies between models and is dependent on the amount of GPU ram required per thread. Small operations on typical GPUs may see as many as 2000 threads running in parallel. Meanwhile even the largest cloud GPUs max out around 80GB of on board memory. If argon2 is configured to use 1GB to compute password hashes then even an NVIDIA A100 would only be able to try 80 passwords in parallel instead of the orders of magnitude more cores that would be active when attacking PBKDF2 hashed passwords.

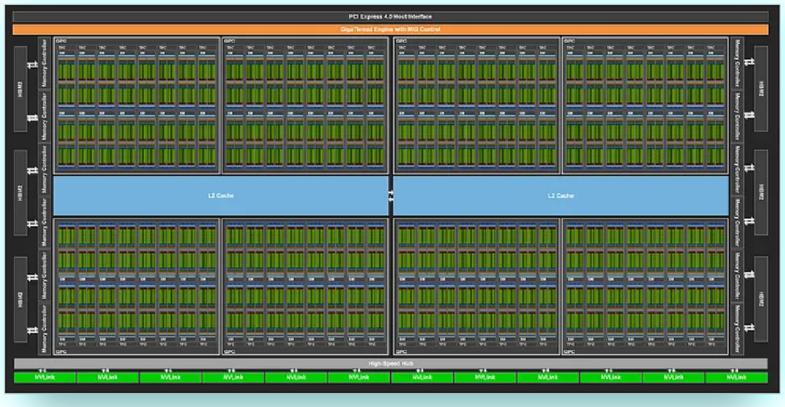

PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
L2 Cache
L2 Cache
High-Speed Hub
NVLink
Memory Controller
HBM2

Diagram of NVIDIA A100 GPU

Argon2 And You . . .

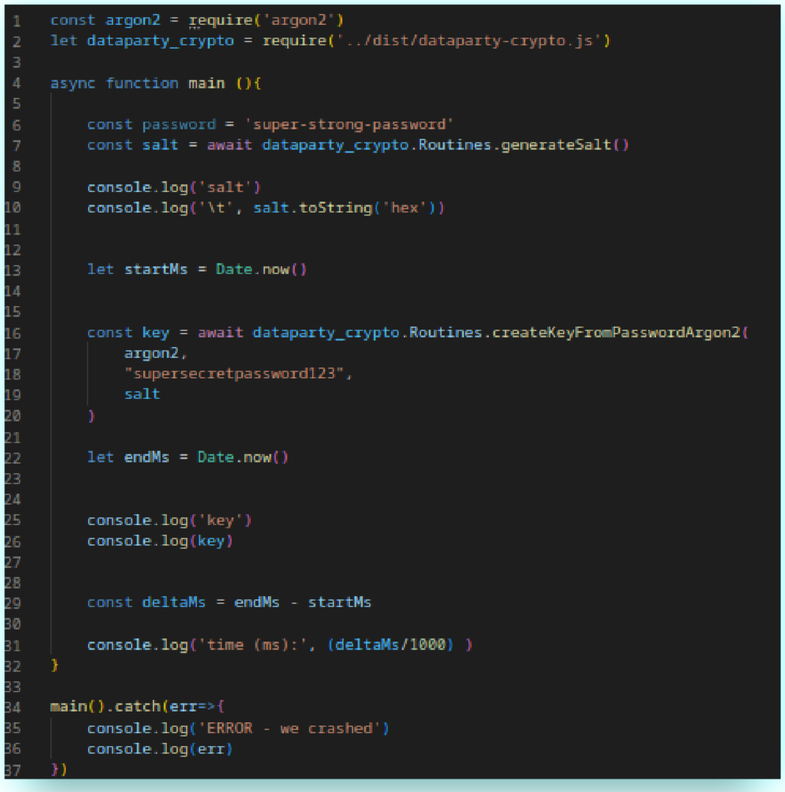
If you’re a developer who builds secure apps and this is your first time hearing about argon2 its probably a good time to review your code. On nodejs check for uses of crypto.pbkdf2 that should be upgraded.

If you’re a Linux user, more likely than not your LUKS partitions will already using `argon2id`. You can check by running the `lsblk` command to find the name of an active partition. Then running:

```
sudo cryptsetup luksDump /dev/<YOUR_PARTITION_HERE>
```

If you do happen to be using an outdated algorithm you should update it! Matthew Garrett, a Linux developer, has written a great guide to updating old LUKS partitions.

Using Argon2 In Nodejs



```
const argon2 = require('argon2')
let dataparty_crypto = require('../dist/dataparty-crypto.js')

async function main (){

    const password = 'super-strong-password'
    const salt = await dataparty_crypto.Routines.generateSalt()

    console.log('salt')
    console.log('\t', salt.toString('hex'))


    let startMs = Date.now()


    const key = await dataparty_crypto.Routines.createKeyFromPasswordArgon2(
        argon2,
        "supersecretpassword123",
        salt
    )

    let endMs = Date.now()


    console.log('key')
    console.log(key)


    const deltaMs = endMs - startMs

    console.log('time (ms):', (deltaMs/1000) )
}

main().catch(err=>{
    console.log('ERROR - we crashed')
    console.log(err)
})
```

Example argon2 password KDF

At the hacker collective, dataparty, we've been building a secure configuration feature that we intend to use to make a secure, decentralized database management tool. We'd recently relied upon pbkdf2 but with the news of Ivan's letter from a French prison we've taken the time to upgrade to argon2 in our nodejs codebases.

You can see how we upgraded to argon2 by reading through this feature request and the PRs referenced within.

Nodejs Vs. Browser

We had some trouble getting the nodejs focused module `argon2` and the browser module `argon2-browser` to play nice together. Sadly these libraries do not both use the same API. We made a wrapper function in `@dataparty/crypto` that allows you to use the same API for both modules. We've posted a complete example for nodejs usage on github:

`dataparty-crypto/example-password-argon2.js` at master · `@dataparty/crypto`

Meanwhile we've added an argon2 example to our browser example.

Support

Find this story helpful? Buy us a coffee or give a follow:

- https://ko-fi.com/dataparty
- https://github.com/datapartyjs

- https://dataparty.xyz
- Join our discord https://discord.gg/JrYQ3f4Pxz

THE LATEST >>