“前端算力攻防”

2017.03 | @EtherDream

分享内容

本次分享主要内容

- 算力攻击
- 算力防御
- 计算优化

前端能做什么

先来想象下：

假如黑客控制了一个访问量巨大的网页，

那么，可以用它来做些什么？

前端能做什么

传统的利用方式

- 挂马
- 打广告
- 界面钓鱼
- 隐私收集

前端能做什么

网络资源也能利用

- 内网扫描
- CC / DDOS （Great Cannon）

前端能做什么

硬件资源，是否也能利用?

- CPU
- GPU
- 内存
- ...

算力攻击

纯粹的计算，也有很多玩法

- 在线挖矿
- 密码破解
- ...

计算的价值

工作量证明

只能通过 暴力穷举 才能解决的问题

```
MD5(X) == X
```

计算很困难，验证却很容易。

工作量证明

难度控制

```
Hash(问题 + 答案) == '000000……'
```

- 条件：结果前 N 位都是 0
- 效果：N 越大，难度越大

工作量证明

JavaScript PoW 演示

http://www.etherdream.com/FunnyScript/hashcash/js/test.html

http://www.etherdream.com/hashcash/login.php

https://www.cnblogs.com/index-html/p/web-pow-research.html

口令破解

口令 Hash 破解，也需要大量的计算

```
b7a2cbd7bcce4b87 == hash(?)
```

Hash 函数不可逆，所以只能穷举。

口令破解

暴力穷举:

```
for i = 0 … 1000000000
    if hash(i) == b7a2cbd7bcce4b87
        cracked !
```

字典穷举:

```
for i in dict
    if hash(i) == b7a2cbd7bcce4b87
        cracked !
```

前端计算

相比客户端，浏览器面临更多挑战

- 如何提高脚本性能?
- 如何充分利用硬件?
- 如何延长运行时间?

性能提升

HTML

- WebAssembly / asm.js

Flash

- FlasCC、HaXe

CPU 潜力

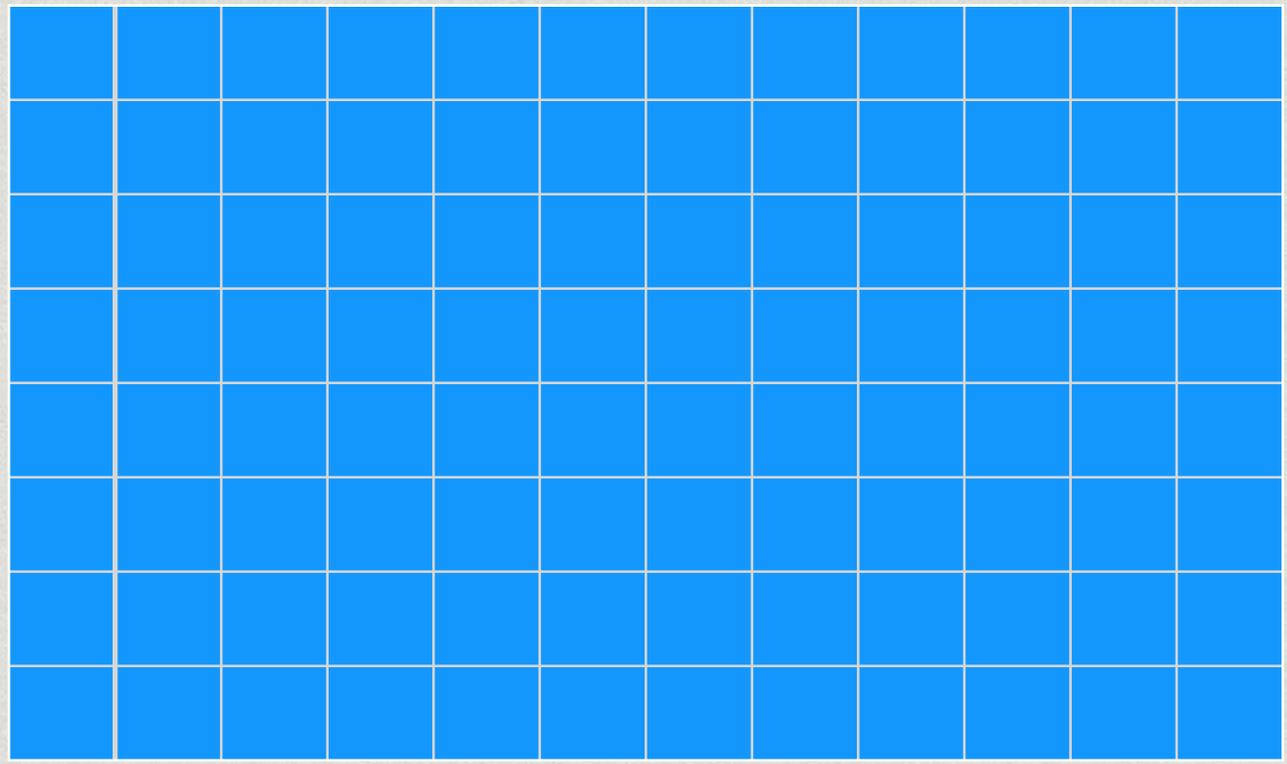
通过 Worker，充分利用 CPU 资源

```
for i = 0 … navigator.hardwareConcurrency
    new Worker(url)
```

GPU 潜力

CPU：

GPU：



GPU 潜力

WebGL API

- 初衷

  通过 GPU 加速 3D 图形渲染

- 利用

  通过 GPU 加速 天生可并行 的计算

通用计算

GP-GPU (General Purpose GPU)

* <canvas height="1" width="**并发数**">

- **输入**：uniform / attribute / 纹理
- **算法**：GLSL 着色器
- **输出**：片元 RGBA 颜色值

WebGL2

支持 整数、位运算，密码学算法的基础

演示：

- SHA256 PoW

  http://www.etherdream.com/FunnyScript/glminer/glminer.html

- WiFi Crack (WPA-2)

  http://127.0.0.1:8080/wpa2-test.html

GLSL 优化

运行时常量替换:

```
var shaderCode = `
    salt = ${SALT}
    saltLen = ${SALT_LEN}
    ……
    if (final_hash == ${HASH}) {
        // cracked
    }
`
```

计算连续性

页面关闭、刷新、跳转，计算就会中断

改进方案：

- SharedWorker
- SharedWorker
- Broadcast Channel API

计算持久性

离开受控页面，计算会就终止

改进方案：

- ServiceWorker
- XSS 续命魔法（注入 opener、劫持超链接）

  演示：http://www.etherdream.com/FunnyScript/XSSGhost/

方案对比

WebAssembly

- 设备：CPU
- 算力：中
- 体验：无感知
- 可在 Worker 中运行

WebGL2

- 设备：GPU
- 算力：高
- 体验：界面有卡顿
- 只能在页面运行

算力防御

口令破解

为什么 Hash 后的口令 仍能破解?

- 口令大多有规律，可以跑字典
- 计算时间短，所以跑字典很快

所以，需要 提高 Hash 计算时间

慢 Hash

高成本的 Hash 函数，可以增加破解时间

- PBKDF2
- bcrypt
- scrypt
- argon2

前端计算

口令 Hash 在前端计算，分担后端压力

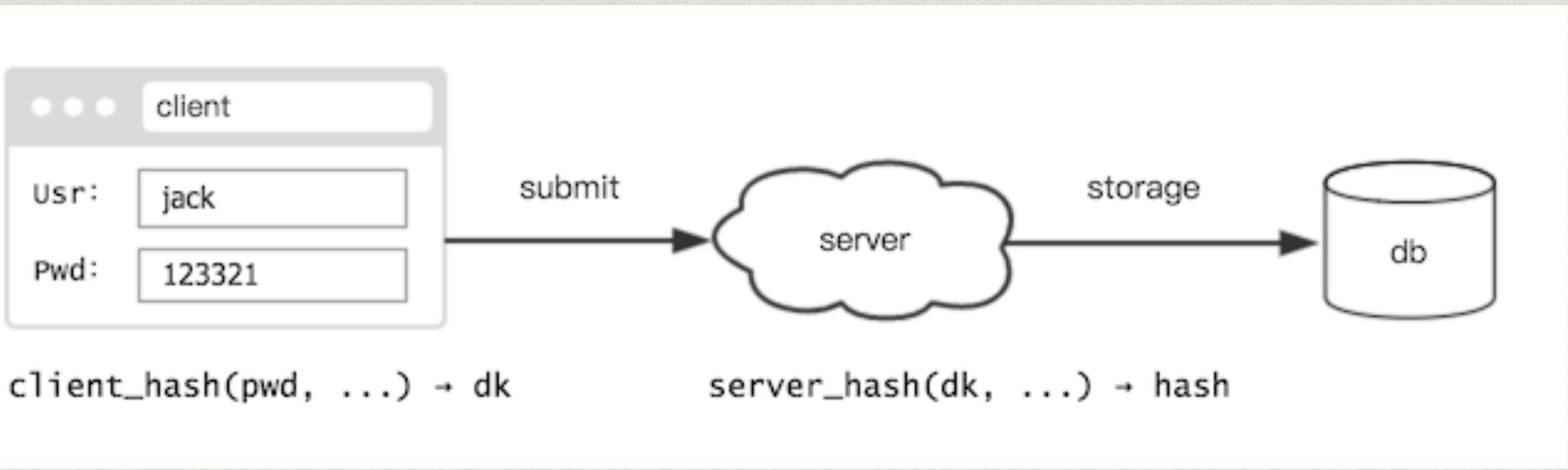
```
dk = hash(password, username, cost …)
```

- 注册时，使用 dk 代替口令
- 登录时，使用同样算法，算出 dk

如果 dk 相同，意味口令相同（无需提交明文口令）

前端计算

后端收到后，再 hash 一次，防止 dk 泄露



其他意义

前端 Hash 除了分担后端计算，还可以：

- 增加登录成本，对抗撞库攻击
- 明文更早消失，减少泄露环节

演示

scrypt hash 算法前端计算

http://www.etherdream.com/webscrypt/example/login/

https://github.com/EtherDream/webscrypt

改进

使用 WebGL2 强化口令 Hash

线程 1：R[1] = KDF(password, salt + ‘1’, cost …)

线程 2：R[2] = KDF(password, salt + ‘2’, cost …)

…

线程 1024：R[1024] = KDF(password, salt + ‘1024’, cost …)

最终结果：DK = Hash(R[1] + R[2] + … + R[1024])

分享总结

- 算力攻击（免费矿工、口令破解）
- 算力防御（PoW 风控、前端 Hash 口令）
- 计算优化（CPU、GPU、连续性、持久性）