【不就是浏览器挖矿嘛】Coinhive挖矿脚本分析与Pool改造自建(二)

evil7 / 2017-10-20 04:34:00 / 浏览数 4921 技术文章 技术文章 顶(0) 踩(0)

自己动手丰衣足食

本着一探究竟开源共享的精神，朝着拿回我的30%payback目标，我们已经详细分析了Coinhive挖矿脚本的构成、由来、运作方式，暂不提用户交互和兼容处理方式，我们

转化器思路构想

1. 从现有开源的矿池项目直接二次构造，搭建兼容WebSocket通信方式的完整Pool。

   优点：从矿池整体可控，全面覆盖各项设置，100%赚取算力价值，矿池直接收取价值。（一般Pool由中心矿池打款到矿工需满足至少有0.1XMR，其中自动“税收”扣去矿

   缺点：服务器配置需求较高（≥2C4G），对于小流量站点或者前端产品变现转换率较低，运营赤字风险大。其次矿池更新迭代需从原项目升级并重新修改，容易出现不必

1. 从头构造流量转换，用 `Pool_Proxy` 形式，对接转化WebSocket与PoolSocket，以中间件形式介入。

   优点：只需构造中间件，便于维护。可以单独形成Log，对搭建平台要求无过高要求。

   缺点：无法从头操控，无法避免部分定量捐赠和Pool平台“税收”。

从优缺点看，我们首要选择从 `Pool_Proxy`
中间件方式来构造前端挖矿的服务端，而后端Pool的选择空间就更大了，可以选择现有的公开矿池，也可以另外结合再自己搭建矿池。

稍安勿躁，本文将分别讲解自建中间件的过程以及标准矿池的搭建方式。

中间件deepMiner构造

为了简化开发流程，我使用比较熟悉的nodejs举例实现（其他语言按需实现均可）。

中间件制作，用于转化WebSocket流量与PoolSocket(TCP)流量，给双方充当“翻译”角色。

所以基础框架，先获得两边的接口，并使其能够正常对接，所以我们需要先写入如下内容到一个新建的 `server.js` 里：

```
var http = require('http'),     //web■■
   WebSocket = require("ws"), //WebSocket■■
   net = require('net'),      //PoolSocket(TCP)■■
   fs = require('fs');        //■■■■■■■■
```

我们先构造一个 `config.json` 文件用来设置构造所需的参数设定，比如域名地址，矿池地址，钱包地址，监听端口等：

```
{
   "lhost": "127.0.0.1",
   "lport": 7777,
   "domain": "miner.deepwn.com",
   "pool": "pool.usxmrpool.com:3333",
   "addr": "41ynfGBUDbGJYYzz2jgS****************************************************",
   "pass": ""
}
```

再继续往 `server.js` 里加入代码，读取配置文件并构造出大体框架。

先来一个web，确保外部访问正常：

```
var conf = fs.readFileSync(__dirname + '/config.json', 'utf8');
conf = JSON.parse(conf);
// Http web
var web = http.createServer((req, res) => {
   res.setHeader('Access-Control-Allow-Origin', '*');
   res.end('Pool Worked!'); // Change at Next...
   }).listen(conf.lport, conf.lhost);

// next codes here...
```

打开浏览器，从自己的127.0.0.1:7777已经能访问看到`Pool Worked!`页面。

接下来完成复用构造WebSocket服务：

为了方便书写，我们先声明一个叫做 conn 的对象来接管所有设置内容，为了方便后期调用。

其中囊括了ws服务，以及每次ws新连接所触发的`net.Socket()`，同时声明每个连接的 `pid` 来解决一个关于`Miner_banned`的坑……

回顾一下：上篇稿件中提到的JsonRPC里，首次login验证中的id，其实是一个Miner的身份区分。

```
client >>

{
   "method": "login",
   "params": {
       "login": "********** [ Wallet Addr ] **********",
       "pass": "",
       "agent": "xmr-stak-cpu/1.3.0-1.5.0"
   },
   "id": 1 // <= ■■id ■■■■■■■■■■■■■■■■■■jobs■■■■■■id=1■■■■■■■■Miner■ban■
}

server <<

{
   "id": 1, // <= ■■■MinerID■■■IP■■■■■■■■■DHCP■■Miner■■IP■■■■■■■……
   "jsonrpc": "2.0",
   "error": null,
   "result": {
       "id": "811233385116793",
       "job": {
           "blob": "0606e498c5ce057326423f235dcd67dec07d9cb79e3506da8b35198e7debb40be3cbc2326c1999000000008bad7c9d5b78e9c96
           "job_id": "664084446453489",
           "target": "711b0d00"
       },
       "status": "OK"
   }
}
```

这是pool用来区分单个IP不同miner的MinerID，如果一个IP里同一个Miner多次违约不完成Job并且重复登录申请新Job，将会进入banned模式，10分钟内无法获取新Jobs

因为只是个demo，所以所有内容，全写在一个文件了，并没有进行区分和不同Socket线程单独控制，就全权交给`http`来内部控制sessions开启和销毁吧！我们继续接着构

```
// Websocket ■■■■■■■■■■■■TCP_Socket■■Pool■■■■■TCP■■
var srv = new WebSocket.Server({
   server: web, // ■■■web■■ws■■■
   path: "/proxy", //■■■■path■■
   maxPayload: 256
});
srv.on('connection', (ws) => { //■■■■■■■■■■■■■conn
   var conn = {
       uid: null, //■■■■■■■■■■■■■■■■UID
       pid: new Date().getTime(), //■■■■……■■MinerID
       workerId: null, //■■PoolJobs■job_id
       found: 0,
       accepted: 0,
       ws: ws, //this ws
       pl: new net.Socket(), //TCP Socket
   }
   var pool = conf.pool.split(':');
   conn.pl.connect(pool[1], pool[0]); //■■■conn.pl■■■TCPSocket■■Pool

   // on.('event') & some func here...

});
```

我们可以 `nc -lvvp 8888` 本地监听，修改 `config.json` 里pool的地址为本地监听的接口，再通过浏览器构造WebSocket访问 `ws://127.0.0.1:7777` 来验证代码是否可以执行。

在一切顺利的情况下我们开始下一步，处理不同事件，现在可以将 `// on.('event') & some func here...` 改为：

```
// Trans func here...

conn.ws.on('message', (data) => {
   ws2pool(data); // Trans WS2TCP
   console.log('[>] Request: ' + conn.uid + '\n\n' + data + '\n');
});
conn.ws.on('error', (data) => {
   console.log('[!] ' + conn.uid + ' WebSocket ' + data + '\n');
   conn.pl.destroy();
});
conn.ws.on('close', () => {
   console.log('[!] ' + conn.uid + ' offline.\n');
   conn.pl.destroy();
});
conn.pl.on('data', (data) => {
   pool2ws(data); // Trans TCP2WS
   console.log('[<] Response: ' + conn.uid + '\n\n' + data + '\n');
});
conn.pl.on('error', (data) => {
   console.log('[!] PoolSocket ' + data + '\n');
   if (conn.ws.readyState !== 3) {
       conn.ws.close();
   }
});
conn.pl.on('close', () => {
   console.log('[!] PoolSocket Closed.\n');
   if (conn.ws.readyState !== 3) {
       conn.ws.close();
   }
});
```

`conn.ws.on('event', [function])` 接管了在不同情况下对WebSocket的处理方式。

`conn.pl.on('event', [function])` 接管了对接Pool的不同处理方式。

那么我们还少了什么？ 对，如何转换Socket流量才是核心内容，我们替换刚才 `// Trans func here...` 为如下，开始勾画核心——Socket转换的Functions：

```
// Trans WebSocket to PoolSocket
function ws2pool(data) {
   var buf;
   data = JSON.parse(data);
   switch (data.type) {
       case 'auth':
           {
               conn.uid = data.params.site_key;
               if (data.params.user) {
                   conn.uid += '@' + data.params.user;
               }
               buf = {
                   "method": "login",
                   "params": {
                       "login": conf.addr,
                       "pass": conf.pass,
                       "agent": "deepMiner"
                   },
                   "id": conn.pid
               }
               buf = JSON.stringify(buf) + '\n';
               conn.pl.write(buf);
               break;
           }
       case 'submit':
           {
               conn.found++;
               buf = {
                   "method": "submit",
                   "params": {
                       "id": conn.workerId,
                       "job_id": data.params.job_id,
                       "nonce": data.params.nonce,
                       "result": data.params.result
                   },
                   "id": conn.pid
               }
               buf = JSON.stringify(buf) + '\n';
               conn.pl.write(buf);
               break;
           }
   }
}

// Trans PoolSocket to WebSocket
function pool2ws(data) {
   var buf;
   data = JSON.parse(data);
   if (data.id === conn.pid && data.result) {
       if (data.result.id) {
           conn.workerId = data.result.id;
           buf = {
               "type": "authed",
               "params": {
                   "token": "",
                   "hashes": conn.accepted
               }
           }
           buf = JSON.stringify(buf);
           conn.ws.send(buf);
           buf = {
               "type": 'job',
               "params": data.result.job
           }
           buf = JSON.stringify(buf);
           conn.ws.send(buf);
       } else if (data.result.status === 'OK') {
           conn.accepted++;
           buf = {
               "type": "hash_accepted",
               "params": {
                   "hashes": conn.accepted
               }
           }
           buf = JSON.stringify(buf);
           conn.ws.send(buf);
       }
   }
   if (data.id === conn.pid && data.error) {
       if (data.error.code === -1) {
           buf = {
               "type": "banned",
               "params": {
                   "banned": conn.pid
               }
           }
       } else {
           buf = {
               "type": "error",
               "params": {
                   "error": data.error.message
               }
           }
       }
       buf = JSON.stringify(buf);
       conn.ws.send(buf);
   }
   if (data.method === 'job') {
       buf = {
           "type": 'job',
           "params": data.params
       }
       buf = JSON.stringify(buf);
       conn.ws.send(buf);
   }
}
```

查看第一篇文章提到的 `coinhive.min.js` 我们可以看到WebSocket主要有 `auth` / `submit` / (`banned`) 三个不同内容。

而从Socket_Dump中我们看到，PoolSocket里只有标准化的 `JsonRPC` ，所以我们需要转换出当前脚本能接受的 `authed` / `job` / `hash_accepted` / (`error`) 四种类型返回如上。

接下来，我们来解决静态资源问题，也是为什么我们要设置 `config.json`
中的域名等，我们需要动态替换所有静态文件里的域名为自己的服务器地址或个人域名，并提供 `cryptonight.wasm`
等其他资源访问，所以我们来修改第一段代码，其中构造的 `web` 实例里需要替换那个 "撒fufu的"页面，将 `res.end('Pool Worked!'); // Change at Next...` 替换为如下：

```
req.url = (req.url === '/') ? '/index.html' : req.url;
   fs.readFile(__dirname + '/web' + req.url, (err, buf) => {
       if (err) {
           fs.readFile(__dirname + '/web/404.html', (err, buf) => {
               res.end(buf);
           });
       } else {
           if (!req.url.match(/\.wasm$/) && !req.url.match(/\.mem$/)) {
               buf = buf.toString().replace(/%deepMiner_domain%/g, conf.domain);
           } else {
               res.setHeader('Content-Type', 'application/octet-stream');
           }
           res.end(buf);
       }
   });
```

将所需的web文件，放入web文件夹，其中lib文件夹放入 `cryptonight-asmjs.min.js` / `cryptonight-asmjs.min.js.mem` / `cryptonight.wasm` ，最终我们拥有了一个200行代码写出来的 `Pool_Proxy` 中间件！

deepMIner 项目实例

参考Repo: https://github.com/deepwn/deepMiner

Example: https://deepc.cc/demo.html

```
deepMiner.git
.
|-- README.md
|-- banner
|-- config.json
|-- package-lock.json
|-- package.json
|-- server.js
|__ web
   |-- 404.html
   |-- deepMiner.js
   |-- demo.html
   |-- index.html
   |-- lib
   |   |-- cryptonight-asmjs.min.js
   |   |-- cryptonight-asmjs.min.js.mem
   |   |__ cryptonight.wasm
   |__ worker.js
```

构建属于自己的Pool

（以下内容，可以跳过或者选取阅读。你可以直接在 `config.json`
里使用对外开放的公共矿池，也可以继续跟着本文，搭建自己的矿池。因为这个200行的Pool_Proxy已经可以完美地独立运转了！）

既然中间件有了，按道理我们可以直接使用，但还是想自行控制全部权限。

所以，不如再来一起搭建一个完全属于自己的矿池吧！

Github: https://github.com/zone117x/node-cryptonote-pool

Monero: https://getmonero.org

搭建Monero

首先，保证我们的服务器有 `1C1G` 的标准，因为本矿池并不对外，可以只在localhost运行，所以我们不需要太大的规格来容纳那么多连接。

但是为了确保万一别用着用着就宕了……我们还是先设置一下虚拟内存吧……

```
dd if=/dev/zero of=/mnt/myswap.swap bs=1M count=4000
mkswap /mnt/myswap.swap
swapon /mnt/myswap.swap
```

再把设置出来的内存，挂载到系统里 `vi /etc/fstab` 并加入如下（保存退出： `:wq` ）

```
/mnt/myswap.swap none swap sw 0 0
```

接下来解决Pool的依赖问题：

```
apt-get install build-essential libtool autotools-dev autoconf pkg-config libssl-dev
apt-get install libboost-all-dev git libminiupnpc-dev redis-server
add-apt-repository ppa:bitcoin/bitcoin
apt-get update
apt-get install libdb4.8-dev libdb4.8++-dev
```

麻烦的事情总是要来的，我们需要得到完整版区块链信息来完成交易和任务发布，所以需要构建可信的 `monerod` 本地进程。

具体Monero版本如果更新了，可以去官网下载布置，本文目前以0.11.0.0版本介绍。（见上方链接）

```
cd
mkdir monero
cd monero
wget https://downloads.getmonero.org/cli/monero-linux-x64-v0.11.0.0.tar.bz2
tar -xjvf monero-linux-x64-v0.11.0.0.tar.bz2
```

然后运行 `./monerod` 开始长达3-6小时的下载区块链信息和验证完整性……

当然，官网也介绍了一个更方便的方式，直接手动下载 `raw` 文件并导入验证。

```
// ■■■■■■■
wget -c --progress=bar https://downloads.getmonero.org/blockchain.raw
./monero-blockchain-import --verify 0 --input-file ./blockchain.raw

// ■■■■■■■■■■■~
rm -rf ./blockchain.raw

// ■■■■demon■■■■■
./monerod --detach
```

搭建Pool

可以参见官方文档：https://github.com/zone117x/node-cryptonote-pool#1-downloading--installing

之前已经下载了 `nodejs` 所以不做重复下载。我们还需要 `Redis` 解决Pool的数据库问题。

关于Redis安装，一搜一大堆，不再啰嗦。

> 请注意：切记设置Redis为本地服务，不要对外开放，可以自行设置密码。
>
> 参见：https://redis.io/topics/security

搭建完Redis数据库，我们来从github下载最新的源码，着手布置Pool。

```
cd /srv
git clone https://github.com/zone117x/node-cryptonote-pool.git pool
cd pool
npm update
```

等npm更新下载完毕，我们来进行配置

```
cp config_example.json config.json
```

`vi config.json` 开始配置Pool信息

```
/* Used for storage in redis so multiple coins can share the same redis instance. */
"coin": "monero",
/* Used for front-end display */
"symbol": "MRO",

"logging": {

   "files": {

       /* Specifies the level of log output verbosity. This level and anything
          more severe will be logged. Options are: info, warn, or error. */
       "level": "info",

       /* Directory where to write log files. */
       "directory": "logs",

       /* How often (in seconds) to append/flush data to the log files. */
       "flushInterval": 5
   },

   "console": {
       "level": "info",
       /* Gives console output useful colors. If you direct that output to a log file
          then disable this feature to avoid nasty characters in the file. */
       "colors": true
   }
},

/* Modular Pool Server */
"poolServer": {
   "enabled": true,

   /* Set to "auto" by default which will spawn one process/fork/worker for each CPU
      core in your system. Each of these workers will run a separate instance of your
      pool(s), and the kernel will load balance miners using these forks. Optionally,
      the 'forks' field can be a number for how many forks will be spawned. */
   "clusterForks": "auto",

   /* Address where block rewards go, and miner payments come from. */
   "poolAddress": "4AsBy39rpUMTmgTUARGq2bFQWhDhdQNek******************************************************************"

   // ■■■■■■■■■■■■■Wallet■■

   /* Poll RPC daemons for new blocks every this many milliseconds. */
   "blockRefreshInterval": 1000,

   /* How many seconds until we consider a miner disconnected. */
   "minerTimeout": 900,

   // ■■■■■■web■■■■■■■■■■■■■■■■■■■■
   "ports": [
       {
           "port": 1111, //Port for mining apps to connect to
           "difficulty": 100, //Initial difficulty miners are set to
           "desc": "Low end hardware" //Description of port
       },
       {
           "port": 2222,
           "difficulty": 500,
           "desc": "Mid range hardware"
       },
       {
           "port": 3333,
           "difficulty": 1000,
           "desc": "High end hardware"
       }
   ],

   /* Variable difficulty is a feature that will automatically adjust difficulty for
      individual miners based on their hashrate in order to lower networking and CPU
      overhead. */
   "varDiff": {
       "minDiff": 2, //Minimum difficulty
       "maxDiff": 1000,
       "targetTime": 100, //Try to get 1 share per this many seconds
       "retargetTime": 30, //Check to see if we should retarget every this many seconds
       "variancePercent": 30, //Allow time to very this % from target without retargeting
       "maxJump": 100 //Limit diff percent increase/decrease in a single retargetting
   },

   /* Feature to trust share difficulties from miners which can
      significantly reduce CPU load. */
   "shareTrust": {
       "enabled": true,
       "min": 10, //Minimum percent probability for share hashing
       "stepDown": 3, //Increase trust probability % this much with each valid share
       "threshold": 10, //Amount of valid shares required before trusting begins
       "penalty": 30 //Upon breaking trust require this many valid share before trusting
   },

   /* If under low-diff share attack we can ban their IP to reduce system/network load. */
   "banning": {

       //banning■■■■■false■■■web■■■■■■■■■■■■■■■■■■■■
       //■■■■■■■■■■■■■■■■■■■■PoolProxy■■■■■■30■■■■■■■■■

       "enabled": true,
       "time": 30, //How many seconds to ban worker for
       "invalidPercent": 25, //What percent of invalid shares triggers ban
       "checkThreshold": 30 //Perform check when this many shares have been submitted
   },
   /* [Warning: several reports of this feature being broken. Proposed fix needs to be tested.]
       Slush Mining is a reward calculation technique which disincentivizes pool hopping and rewards
       'loyal' miners by valuing younger shares higher than older shares. Remember adjusting the weight!
       More about it here: https://mining.bitcoin.cz/help/#!/manual/rewards */
   "slushMining": {
       "enabled": false, //Enables slush mining. Recommended for pools catering to professional miners
       "weight": 300, //Defines how fast the score assigned to a share declines in time. The value should roughly be equivalen
       "lastBlockCheckRate": 1 //How often the pool checks the timestamp of the last block. Lower numbers increase load but ra
   }
},

/* Module that sends payments to miners according to their submitted shares. */
"payments": {
   "enabled": true,
   "interval": 600, //how often to run in seconds
   "maxAddresses": 50, //split up payments if sending to more than this many addresses
   "mixin": 3, //number of transactions yours is indistinguishable from
   "transferFee": 5000000000, //fee to pay for each transaction
   "minPayment": 100000000000, //miner balance required before sending payment
   "denomination": 100000000000 //truncate to this precision and store remainder
},

/* Module that monitors the submitted block maturities and manages rounds. Confirmed
  blocks mark the end of a round where workers' balances are increased in proportion
  to their shares. */
"blockUnlocker": {
   "enabled": true,
   "interval": 30, //how often to check block statuses in seconds

   /* Block depth required for a block to unlocked/mature. Found in daemon source as
      the variable CRYPTONOTE_MINED_MONEY_UNLOCK_WINDOW */
   "depth": 60,
   "poolFee": 1.8, //1.8% pool fee (2% total fee total including donations)
   "devDonation": 0.1, //0.1% donation to send to pool dev - only works with Monero
   "coreDevDonation": 0.1 //0.1% donation to send to core devs - only works with Monero
},

/* AJAX API used for front-end website. */
```

```
"api": {
   "enabled": true,
   "hashrateWindow": 600, //how many second worth of shares used to estimate hash rate
   "updateInterval": 3, //gather stats and broadcast every this many seconds
   "port": 8117,
   "blocks": 30, //amount of blocks to send at a time
   "payments": 30, //amount of payments to send at a time
   "password": "test" //password required for admin stats
},

/* Coin daemon connection details. */
"daemon": {
   "host": "127.0.0.1",
   "port": 18081
},

/* Wallet daemon connection details. */
"wallet": {
   "host": "127.0.0.1",
   "port": 8082
},

/* Redis connection into. */
"redis": {
   "host": "127.0.0.1",
   "port": 6379,
   "auth": null //If set, client will run redis auth command on connect. Use for remote db
}
```

最后在终端键入 `node init.js` 让Pool开始工作, 也可以用比如 `node init.js -module=api` 只开启单独项目。

当然你也可以用 `forever start /srv/pool/init.js`
确保出错了还能在线，也可以将其写入开机启动项里。不过值得注意的是：切记要把Pool的开启，放在Pool_Proxy开启之前哦！

如此一来，Pool + Pool_Proxy 就完成了，请开始你的表演吧~

Pool 相关项目

https://github.com/zone117x/node-cryptonote-pool

https://github.com/CanadianRepublican/monero-universal-pool

https://github.com/search?utf8=%E2%9C%93&q=monero+pool

发展构想

其实前端算力，不仅仅可以用来挖矿，更可以用来做机器人验证，构造一种算法，或者换一种token方式，利用硬算力来增加批量化成本，同时通过算力难度，来增加单次的

同时 `asmjs` 和 `WebAssembly`
的出现，也将前端的处理能力提升到一个新的台阶，今后通过浏览器构造本地应用？创建新形式的3D页游？甚至加入P2P将应用分发到用户？新的加密传输？大众化的自定

正如Coinhive现在的验证码雏形，市场前景十分美好，通过基本判断，逐步增加重复提交表单的计算成本，来杜绝撸羊毛，通过不断的更新与进步，前端的魅力，正在逐步

比如EtherDream的文章推荐给大家：

使用浏览器的计算力，对抗密码破解

【探索】无形验证码 —— PoW 算力验证

怎样的 Hash 算法能对抗硬件破解

> 一入前端深似海，猥琐不往此生学。
>
> 熊迪，跟我来学做菜吧2333......

点击收藏 | 0 关注 | 0
上一篇：OSSIM（开源安全管理平台）技术... 下一篇：甲方安全建设步骤
1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区
