Protecting Against Domain Front-Running by Registrars

Understanding and Mitigating Domain Registration Interception

Kabui, Charles

2024-10-27

Table of Contents

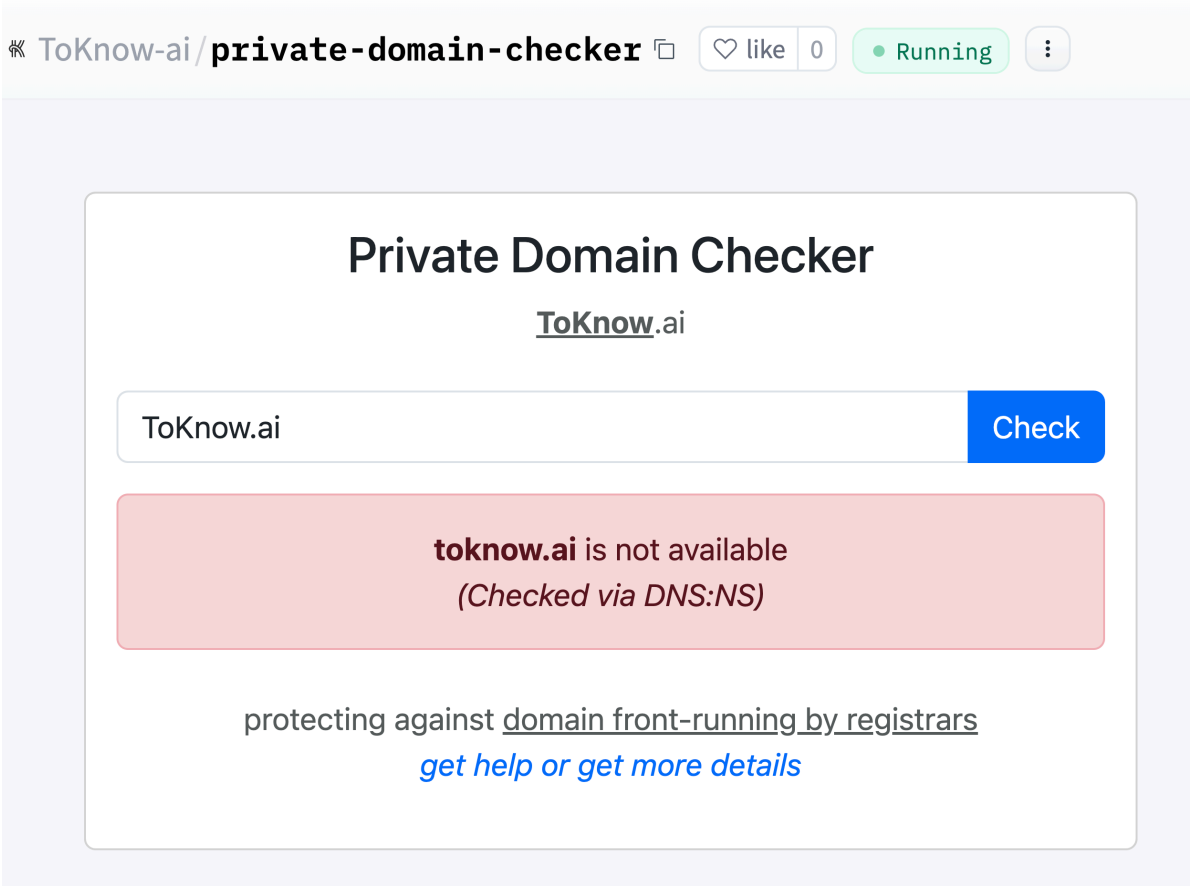

Figure 1: Private Domain Checker

Introduction

Domain front-running[1] typically occurs when a domain registrar or associated party monitors search queries entered by users looking for available domain names. If the domain appears promising or marketable, the registrar may register it before the user, intending to profit by reselling it at a higher price. While registrars deny engaging in this practice, affected individuals and businesses find it challenging to prove due to limited transparency regarding how domain search data is managed and monitored.

On a personal level, I, along with several friends and family members, have encountered domain front-running. In a recent incident, after checking a domain's availability through a Kenyan affiliate registrar, the exact domain was unexpectedly registered within minutes by the primary affiliated registrar. On average, short and simple domain names are typically lost within `12-72 hours` after searching, depending on where you searched and the TLD of the domain.

[1] https://en.wikipedia.org/wiki/Domain_name_front_running

Such incidents are far from isolated. Numerous domain seekers report similar patterns, noting that after searching for specific domains, these names quickly become registered by the same or affiliated registrars. This behavior leaves many users without clear explanations, contributing to widespread suspicion and concern. Evidence from personal experiences and community reports suggests that domain front-running remains an ongoing concern that affects domain seekers.

Old and Recent online discussions across various forums highlight ongoing concerns about domain front-running:

- *Domain Front Running?* - Sunday, August 4, 2024
- *Safe way to search for availability to avoid front running?* - Friday, September 24, 2021
- *Did GoDaddy just front run me?* - Wednesday, April 10, 2024
- *I just confirmed that Namecheap buys available domains that you search for* - Tuesday, March 28, 2023
- Others - https://www.google.com/search?q=domain+front-running+site%3Areddit.com

Understanding Domain Registration Systems

Domain registration involves multiple levels of authority:

- Country-level registries (e.g., KENIC for `.ke` domains)
- Global registries (e.g., Verisign for `.com` and `.net`)
- Individual registrars (e.g., GoDaddy, Namecheap)

Each registry maintains authoritative records of registered and unregistered domains through their `WHOIS` or `RDAP` protocols. Some TLDs face stability issues, such as Libya's `.ly` domain, where the official registrar (whois.nic.ly) is non-functional, leaving only unofficial services like reg.ly.

There are two main protocols for checking domain availability, `WHOIS` and `RDAP`:

Feature	WHOIS Protocol	RDAP (Registration Data Access Protocol)
Introduced	1982	2015
Status	Legacy but widely used	Modern replacement for WHOIS
Privacy Features	Limited	Built-in
Standardization	Varies by registrar	JSON-based, consistent
Accuracy	Can be inconsistent	Generally more reliable

Feature	WHOIS Protocol	RDAP (Registration Data Access Protocol)
Advantages	- Widespread support- Simple protocol	- Standardized JSON responses- Better privacy controls- More efficient queries
Disadvantages	No standardized format- Limited privacy protections- Rate limiting issues	- Not yet universally adopted- Requires more complex implementation

Best Practices for Private Domain Searches

Claims of domain snatching by domain registrars underscore the need for domain seekers to understand the risks and methods for safeguarding their domain searches.

1. Use Official WHOIS Lookup Sites:

   Instead of searching on a registrar's website, consider using a neutral WHOIS lookup service. Many of these services do not track or record searches, reducing the likelihood of domain front-running.

   - ICANN lookup (https://lookup.icann.org/en) for `.com` domains
   - Direct registry websites (e.g., KENIC for `.ke` domains)
   - Avoid commercial registrars for **initial searches**, such as GoDaddy or NameCheap

2. Utilize command-line tools:

   - Use terminal `whois` command (e.g., `whois example.com`). whois program is usually preinstalled in most unix systems. For Windows users, Microsoft offers a WHOIS utility that can be downloaded from Sysinternals site.
   - Minimizes exposure to potential monitoring

3. Time your searches strategically:

   - Only search on commercial registrars when ready to purchase immediately
   - Make sure you test your payment method before searching

4. Research the Registrar's Reputation:

   Before using a registrar, check reviews and forums for any reports or complaints about front-running. Users often share their experiences, which can help identify registrars with questionable practices.

   - Avoid small resellers who may have less secure practices at all costs

- Understand that most resellers ultimately use major registrars' services, thus have less control over the monitoring of the domain searches

5. Use Independent Domain Search Tools:

   - Utilize unaffiliated services like our private domain checker - https://toknow.ai/apps/private-domain-checker
   - Prefer open-source solutions for transparency

Transparent Solution: Private Domain Checker

Private Domain Checker, hosted on Hugging Face Spaces and with publicly available source code, provides a privacy-focused alternative for domain availability checks. Key features include:

- Multiple checking methods (DNS, RDAP, WHOIS)
- No registrar affiliations
- Open-source codebase
- Support for nearly all TLDs

Limitations:

The tool has some constraints:

- May not check certain TLDs due to non-functional official registries
- Platform accessibility issues in some regions due to Hugging Face hosting restrictions e.g., China[2]
- Dependent on registry API availability and response times
- Dependent on Hugging Face Space resources and availability

Demo

Please visit https://toknow.ai/posts/private-domain-checker/index.html#demo to test the demo.

Code

Below is the code running the flask app running the above private domain checker

[2]whois.cnnic.cn is timming out in huggingface: https://www.chinatalk.media/p/hugging-face-blocked-self-castrating

```python
# https://huggingface.co/spaces/ToKnow-ai/private-domain-checker/blob/main/app.py

import json5
import os
import random
from typing import Callable, Literal
from flask import Flask, send_from_directory, request
from urllib.parse import urlparse
import dns.resolver
import socket
import requests
import platform
import subprocess
from shutil import which
import re

app = Flask(__name__)

@app.route('/')
def index():
    """Route handler for the home page"""
    try:
        return send_from_directory('.', 'index.html')
    except Exception as e:
        return str(e)

@app.route('/check', methods=['POST'])
def check():
    return check_domain(request.get_json().get('domain', ''))

def check_domain(domain: str):
    """Check domain availability"""
    logs: list[str] = []
    try:
        domain = validate_and_correct_domain(domain)
        result = check_domain_availability(domain, logs.append)
        if result:
            return {
                "domain": domain,
                "method": f"Checked via {result['method']}",
                "available": result['available'],
                "logs": logs
```

```
            }
        logs.append(f"{check_domain.__name__}:result == None")
    except Exception as e:
        logs.append(f"{check_domain.__name__}:Exception:{str(e)}")
    return default_error(domain, logs)

def validate_and_correct_domain(domain: str):
    # remove leding and trailing "/"
    domain = domain.lower().strip('/').strip()
    # extract domain
    domain = urlparse(domain).netloc.strip() if '://' in domain else domain
    # remove lending non alphanumeric
    while domain and not domain[0].isalnum():
        domain = domain[1:].strip()
    # remove www.
    if domain.startswith("www."):
        domain = domain[4:].strip()
    # remove inner spaces
    domain = re.sub(r'[\n\s]+', '', domain).strip()
    # replace unwanted characters with hyphens
    domain = re.sub(r'[^a-zA-Z0-9\.]', '-', domain).strip('-').strip('.').strip()
    return domain

def default_error(domain: str, logs: list[str]):
    cannot_confirm = "Cannot confirm if doimain is available"
    try:
        current_dir = os.path.dirname(os.path.abspath(__file__))
        with open(os.path.join(current_dir, 'blocked-tlds.jsonc'), mode='r') as f:
            blocked_tlds: list[dict[Literal["tld", "info"], str]] = json5.load(f)
        for blocked_tld in blocked_tlds:
            if domain.endswith(blocked_tld.get('tld')):
                return {
                    'domain': domain,
                    "available": False,
                    "method": f"{cannot_confirm}, try at {blocked_tld.get('info')}",
                    "logs": logs
                }
        response = requests.get("https://data.iana.org/TLD/tlds-alpha-by-domain.txt", timeou
        all_tlds = []
        if response.ok:
            all_tlds = response.text.split("\n")
        else:
```

```
            with open( os.path.join(current_dir, 'tlds-alpha-by-domain.txt'), mode='r') as f
                all_tlds = f.readlines()
        all_tlds: list[str] = [
            i.lower().strip()
            for i
            in all_tlds
            if len((i or '').strip()) > 0 and not i.strip().startswith("#")
        ]
        is_supported_tld = any(True for i in all_tlds if domain.strip().endswith(f'.{i}'))
        if not is_supported_tld:
            return {
                'domain': domain,
                "available": False,
                "method": f"Unsupported domain, \".{'.'.join(domain.split('.')[1:])}\" is not
                "logs": logs
            }
    except Exception as e:
        logs.append(f"{default_error.__name__}:Exception:{str(e)}")
    return {
        'domain': domain,
        "available": False,
        "method": cannot_confirm,
        "logs": logs
    }

def check_domain_availability(domain, logs_append: Callable[[str], None]):
    """Check domain availability using multiple methods."""
    # First try DNS resolution
    is_available, availability_method, _continue = dns_is_available(
        domain, logs_append)
    if not _continue:
        return {
            "available": is_available,
            "method": f"DNS:{availability_method}"
        }

    # Try RDAP
    is_available, availability_method, _continue = rdap_is_available(
        domain, logs_append)
    if not _continue:
        return {
            "available": is_available,
```

```
            "method": f"RDAP:{availability_method}"
        }

    # Fall back to WHOIS
    is_available, availability_method, _continue = whois_is_available(
        domain, logs_append)
    if not _continue:
        return {
            "available": is_available,
            "method": f"WHOIS:{availability_method}"
        }

def dns_is_available(domain, logs_append: Callable[[str], None]):
    """Check if domain exists in DNS by looking for common record types."""
    # Check NS records first as they're required for valid domains
    try:
        resolver = get_dns_resolver()
        for record_type in ['NS', 'A', 'AAAA', 'MX', 'CNAME']:
            try:
                resolver.resolve(domain, record_type)
                return False, record_type, False
            except Exception as e:
                logs_append(
                    (f"{dns_is_available.__name__}:{record_type}:Exception"
                     f":{'|'.join(resolver.nameservers)}:{str(e)}"))
    except Exception as e:
        logs_append(f"{dns_is_available.__name__}:Exception:{str(e)}")
    return True, None, True

def get_dns_resolver():
    # list of major DNS resolvers
    resolver = dns.resolver.Resolver()
    def myshuffle(ls):
        random.shuffle(ls)
        return ls
    namesevers = {
        'cloudflare': myshuffle(['1.1.1.1', '1.0.0.1']),
        'google': myshuffle(['8.8.8.8', '8.8.4.4']),
        'quad9': myshuffle(['9.9.9.9', '149.112.112.112']),
        'opendns': myshuffle(['208.67.222.222', '208.67.220.220']),
        'adguard': myshuffle(['94.140.14.14', '94.140.15.15']),
        'nextdns': myshuffle(['45.90.28.167', '45.90.30.167']),
```

```
        'default': myshuffle(resolver.nameservers)
    }
    resolver.nameservers = random.choice(list(namesevers.values()))
    return resolver

def rdap_is_available(domain, logs_append: Callable[[str], None]):
    try:
        bootstrap_url = "https://data.iana.org/rdap/dns.json"
        bootstrap_data = requests.get(bootstrap_url, timeout=5).json()
        tld = domain.split('.')[-1]
        services: list[tuple[list[str], list[str]]] = bootstrap_data['services']
        for [tlds, rdap_base_urls] in services:
            if tld in tlds:
                for rdap_base_url in rdap_base_urls:
                    response = requests.get(
                        f"{rdap_base_url.lstrip('/')}/domain/{domain}", timeout=5)
                    if response.status_code == 404:
                        return True, rdap_base_url, False
                    elif response.status_code == 200:
                        return False, rdap_base_url, False
        logs_append(f"{rdap_is_available.__name__}:no RDAP")
    except Exception as e:
        logs_append(f"{rdap_is_available.__name__}:Exception:{str(e)}")
    return False, None, True

def whois_is_available(domain, logs_append: Callable[[str], None]) -> bool:
    try:
        available_patterns = [
            'no match',
            'not found',
            'no entries found',
            'no data found',
            'not registered',
            'available',
            'status: free',
            'domain not found',
            'no object found',
            'not been registered',
            'status: available',
            'domain is available',
            'is free',
            'no match found',
```

```
            'domain not registered',
            'domain available',
            'not exists',
            'does not exist',
            'no information available',
            'registration status: unused',
            'status: inactive',
            'no such domain',
            'query matched no objects',
            'no matching record',
            'domain status: available',
            'this domain is not registered',
            'domain name has not been registered',
            'can not find domain',
            'cannot find domain',
            'this domain is available for purchase',
            'domain status: free'
        ]
        is_available_callback = lambda output: any(
            pattern in output for pattern in available_patterns)
        is_available, availability_method = socket_whois_is_available(
            domain, is_available_callback, logs_append)
        if is_available:
            return True, availability_method, False
        is_available, availability_method = terminal_whois_is_available(
            domain, is_available_callback, logs_append)
        if is_available:
            return True, availability_method, False
    except Exception as e:
        logs_append(f"{whois_is_available.__name__}:Exception:{str(e)}")
    return False, None, True

def socket_whois_is_available(
        domain: str,
        is_available_callback: Callable[[str], bool],
        logs_append: Callable[[str], None]):
    try:
        whois_server = get_whois_server(domain, logs_append)

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(4)
        sock.connect((whois_server, 43))
```

```
        sock.send(f"{domain}\r\n".encode())
        response = sock.recv(4096).decode(errors='ignore')
        sock.close()

        response_lower = response.lower()
        return is_available_callback(response_lower), whois_server
    except Exception as e:
        logs_append(
            f"{socket_whois_is_available.__name__}:whois_server:{whois_server}")
        logs_append(
            f"{socket_whois_is_available.__name__}:Exception:{str(e)}")
    return False, None

def terminal_whois_is_available(
        domain: str,
        is_available_callback: Callable[[str], bool],
        logs_append: Callable[[str], None]):
    try:
        # Check if OS is Linux
        if platform.system().lower() == 'linux':
            if which('whois') is not None:
                # Run whois command with timeout
                process = subprocess.Popen(
                    ['whois', domain],
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE)
                try:
                    stdout, stderr = process.communicate(timeout=10)
                    output = stdout.decode('utf-8', errors='ignore').lower()
                    logs_append(
                        (f"{terminal_whois_is_available.__name__}"
                         f":stderr:{str(stderr.decode(encoding='utf-8'))}"))
                    return is_available_callback(output), "system whois"
                except subprocess.TimeoutExpired as timeout_e:
                    logs_append(
                        (f"{terminal_whois_is_available.__name__}"
                         f":TimeoutExpired:{str(timeout_e)}"))
                    process.kill()
            else:
                logs_append(
                    (f"{terminal_whois_is_available.__name__}"
                     ":Exception:WHOIS not installed. "
```

```
                    "Install with: sudo apt-get install whois"))
        else:
            logs_append(
                (f"{terminal_whois_is_available.__name__}"
                ":Exception:System WHOIS check only available on Linux"))
    except Exception as e:
        logs_append(
            f"{terminal_whois_is_available.__name__}:Exception:{str(e)}")
    return False, None

def get_whois_server(domain, logs_append: Callable[[str], None]):
    """Get WHOIS server from IANA root zone database."""
    try:
        response = requests.get(f'https://www.iana.org/whois?q={domain}')
        if 'whois:' in response.text.lower():
            for line in response.text.split('\n'):
                if 'whois:' in line.lower():
                    return line.split(':')[1].strip()
    except Exception as e:
        logs_append(f"{get_whois_server.__name__}:Exception:{str(e)}")
    return None
```

Test

```
check_domain("examplerhccvu.ly")
```


```
{
  "domain": "examplerhccvu.ly",
  "method": "Checked via RDAP:https://rdap.nic.ly/",
  "available": true,
  "logs": [
    "dns_is_available:NS:Exception:208.67.222.222|208.67.220.220:The DNS query name does not
    "dns_is_available:A:Exception:208.67.222.222|208.67.220.220:The DNS query name does not e
    "dns_is_available:AAAA:Exception:208.67.222.222|208.67.220.220:The DNS query name does no
    "dns_is_available:MX:Exception:208.67.222.222|208.67.220.220:The DNS query name does not
    "dns_is_available:CNAME:Exception:208.67.222.222|208.67.220.220:The DNS query name does n
  ]
}
```

```
check_domain("examplerhccvu.cn")
```


```
{
  "domain": "examplerhccvu.cn",
  "method": "Checked via WHOIS:whois.cnnic.cn",
  "available": true,
  "logs": [
    "dns_is_available:NS:Exception:8.8.8.8|8.8.4.4:The DNS query name does not exist: example
    "dns_is_available:A:Exception:8.8.8.8|8.8.4.4:The DNS query name does not exist: examplen
    "dns_is_available:AAAA:Exception:8.8.8.8|8.8.4.4:The DNS query name does not exist: examp
    "dns_is_available:MX:Exception:8.8.8.8|8.8.4.4:The DNS query name does not exist: example
    "dns_is_available:CNAME:Exception:8.8.8.8|8.8.4.4:The DNS query name does not exist: exam
    "rdap_is_available:no RDAP"
  ]
}
```


```
check_domain("examplerhccvu.com")
```


```
{
  "domain": "examplerhccvu.com",
  "method": "Checked via RDAP:https://rdap.verisign.com/com/v1/",
  "available": true,
  "logs": [
    "dns_is_available:NS:Exception:8.8.4.4|8.8.8.8:The DNS query name does not exist: example
    "dns_is_available:A:Exception:8.8.4.4|8.8.8.8:The DNS query name does not exist: examplen
    "dns_is_available:AAAA:Exception:8.8.4.4|8.8.8.8:The DNS query name does not exist: examp
    "dns_is_available:MX:Exception:8.8.4.4|8.8.8.8:The DNS query name does not exist: example
    "dns_is_available:CNAME:Exception:8.8.4.4|8.8.8.8:The DNS query name does not exist: exam
  ]
}
```


```
check_domain("example.com")
```


```
{
  "domain": "example.com",
  "method": "Checked via DNS:NS",
  "available": false,
  "logs": []
}
```

Conclusion

While domain front-running remains challenging to prove conclusively, the abundance of user experiences and technical evidence suggests the need for private domain checking solutions. Our open-source tool provides one approach to mitigating these risks, though industry-wide changes and improved regulations may be necessary for long-term solutions.

To see a liat of all available domain TLDs, see https://data.iana.org/TLD/tlds-alpha-by-domain.txt

---