

Operation TaxShadow : Multi-Region Tax Phishing & In-Memory Malware Campaign

 cyfirma.com/research/operation-taxshadow-multi-region-tax-phishing-in-memory-malware-campaign

Published On : 2026-06-04



Executive Summary

This report presents a detailed investigation of an Indian tax-themed phishing campaign delivering a sophisticated multi-stage malware framework through a combination of social engineering, phishing infrastructure, and memory-resident malware execution techniques. The campaign begins with a fraudulent tax notification email impersonating an official Indian tax authority, leveraging government branding, urgency-based messaging, and compliance-related threats to manipulate victims into interacting with a malicious phishing website. Victims are subsequently instructed to download a malicious ZIP archive containing three staged payload components: कर विवरण.exe, SbieDll.dll, and SbieDll.bin, which collectively establish the complete infection lifecycle. Initial static analysis identified multiple indicators of suspicious behaviour, including modified PE structures, packed components, debug artifacts, and custom execution logic designed to evade conventional security solutions.

Further reverse engineering revealed a highly modular malware architecture implementing advanced defense-evasion and anti-analysis techniques such as DLL Search Order Hijacking, API Hooking, Token Manipulation, COM callback-based execution, Mutated RC4 payload encryption, Reflective PE Loading, and LLVM-based Control Flow Flattening (CFF). The malware executes primarily in memory, significantly reducing forensic artifacts and making traditional detection approaches considerably more difficult. Additionally, the malware establishes persistent WebSocket-based Command-and-Control (C2) communication, allowing malicious traffic to blend with legitimate network activity. Multiple Chinese-language artifacts and suspicious infrastructure indicators were observed throughout the analysis; however, these findings alone are insufficient for definitive attribution and therefore remain at a moderate level of confidence. Overall, the campaign demonstrates characteristics commonly associated with stealth-focused, modular, and highly sophisticated malware ecosystems.

Introduction

The increasing use of government impersonation and social engineering tactics has become a preferred strategy among threat actors seeking to distribute malware through trusted entities and fear-driven narratives. This report analyses an Indian tax-themed phishing campaign designed to deliver a sophisticated multi-stage malware framework through a structured infection chain involving phishing infrastructure, ZIP-based payload delivery, and memory-resident execution mechanisms. The analysis combines campaign infrastructure investigation, static analysis, reverse engineering, execution flow reconstruction, and command-and-control analysis to provide a comprehensive understanding of the attack lifecycle. The primary objective is to identify the campaign's delivery mechanisms, malware functionality, evasion techniques, and operational behaviour, while providing actionable intelligence that can support threat detection, incident response, and security monitoring activities.

Basic Details

Target Technologies	Windows Operating System
Threat Type	Phishing Campaign
File Types	EXE (Portable Executable), DLL (Dynamic Link Library)
Key Malware Identifiers	“कर विवरण.exe”, SbieDll.dll & SbieDll.bin
Observed First	20 May 2026
Impact	Full Access (Command and Control)
MD5 Hashes	कर विवरण.exe “3a8f6454927b8993aded75de0de2bd00” SbieDll.dll “e83ff54e58f0b295a392c7fc39a7d0de” SbieDll.bin “b498256cb086a6962077cdd6d2f65327”

Key Findings

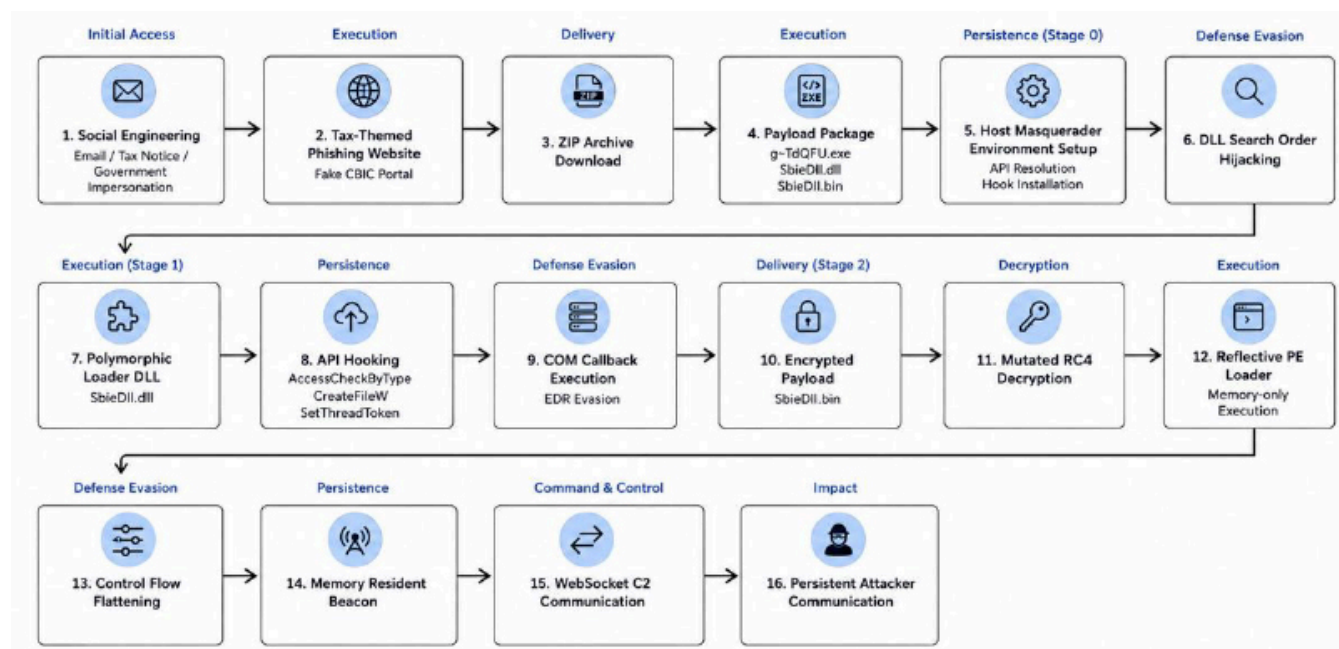
- **Social Engineering-Based Initial Access:** The campaign begins with a fraudulent tax notification email impersonating an official Indian government tax authority, leveraging government branding, bilingual content, and urgency-driven messaging to manipulate victims into interacting with malicious content.
- **Tax-Themed Phishing Infrastructure:** Victims are redirected to a malicious phishing website designed to visually imitate a legitimate tax portal, increasing trust and encouraging payload delivery.
- **ZIP-Based Payload Delivery:** The phishing page delivers a malicious ZIP archive containing multiple staged components responsible for establishing the infection chain.
- **Multi-Stage Payload Architecture:** The archive contains कर विवरण.exe, SbieDll.dll, and SbieDll.bin, forming a structured multi-stage malware execution framework.
- **DLL Search Order Hijacking:** The malware abuses Windows DLL loading behavior to force execution of the malicious SbieDll.dll, enabling unauthorized code execution.
- **Advanced API Hooking Mechanisms:** Multiple hooks, including AccessCheckByType, CreateFileW, DuplicateHandle, SetThreadToken, and GetTokenInformation are implemented to manipulate process behaviour and evade restrictions.
- **Privilege Manipulation and Token Abuse:** The malware performs thread impersonation and token manipulation techniques to bypass access-control mechanisms and execute privileged operations.
- **Custom VM-Based Anti-Analysis Engine:** The malware employs a Mersenne Twister-based execution mechanism that dynamically alters execution characteristics to complicate analysis and signature generation.
- **COM Callback-Based Execution:** Instead of monitored APIs such as CreateThread, malicious code execution is performed through COM IContextCallback handlers, reducing behavioural visibility.
- **Mutated RC4 Payload Encryption:** The shellcode payload uses a modified RC4 stream cipher implementation to protect embedded components and evade traditional detection methods.
- **Reflective PE Injection:** The malware performs memory-resident execution using Reflective PE Loading, eliminating dependency on LoadLibrary() and avoiding file creation on disk.
- **LLVM-Based Control Flow Flattening:** The malware utilizes Control Flow Flattening (CFF) through LLVM-based obfuscation to disrupt normal execution flow and complicate reverse engineering.
- **WebSocket-Based Command-and-Control Communication:** The beacon establishes persistent WebSocket C2 communication using HTTP protocol upgrades, allowing malicious traffic to blend with legitimate network activity.
- **Corporate Proxy Awareness:** Presence of HTTP CONNECT functionality indicates the malware can communicate through enterprise proxy environments, bypassing network restrictions.
- **Memory-Resident Malware Execution:** The malware primarily executes within memory, significantly reducing forensic artifacts and complicating detection by conventional security products.
- **Chinese-Language Artifacts Observed:** Multiple Chinese-language strings and infrastructure indicators were identified during analysis; however, attribution remains at moderate confidence and does not definitively indicate origin.
- **High Sophistication Level:** The combination of anti-analysis techniques, defense evasion, in-memory execution, and stealth-focused communication mechanisms indicates a highly sophisticated and modular malware ecosystem.

Attack Chain

The attack chain begins with a social engineering email impersonating an official Indian tax authority notification, designed to create urgency through financial penalties and compliance-related warnings. The email contains a malicious link directing victims to a fake tax-themed phishing website that visually imitates a legitimate government portal. The phishing page persuades users to download a malicious ZIP archive presented as an official tax notice or supporting document package.

Once extracted, the ZIP archive delivers three staged payload components: कर विवरण.exe, SbieDll.dll, and SbieDll.bin. The executable file acts as the initial execution vector and establishes the environment by performing mutex validation, operating system checks, and dynamic API resolution before installing multiple malicious hooks. The malware then abuses DLL Search Order Hijacking to load the malicious SbieDll.dll instead of a legitimate library. This DLL acts as a polymorphic loader and implements advanced anti-analysis techniques, including API Hooking, Token Manipulation, Mersenne Twister-based execution logic, and COM callback execution for stealth purposes.

The final stage involves loading SbieDll.bin, which contains an encrypted shellcode payload protected using a mutated RC4 stream cipher. Following decryption, the shellcode executes a Reflective PE Loader that manually reconstructs and loads the payload directly into memory without relying on LoadLibrary() or creating artifacts on disk. The malware further protects execution using LLVM-based Control Flow Flattening (CFF) and establishes persistent WebSocket-based Command-and-Control (C2) communication through HTTP protocol upgrades, allowing malicious traffic to blend with legitimate web application activity while maintaining covert communication with attacker-controlled infrastructure.



Analysis

Email Header and Delivery Infrastructure Analysis

The screenshot reveals detailed email header metadata associated with the initial phishing email used during the campaign. Analysis of the header information indicates the use of government impersonation tactics, where the sender identity attempts to imitate an official Income Tax enforcement authority while using a suspicious underlying domain structure. The visible sender address `ac.enforcementincometax.gov.in@mnb-ny.com` demonstrates a display name spoofing technique, where attackers insert a trusted-looking subdomain before the actual controlled domain to mislead recipients into believing the email originates from a legitimate government source.

Further analysis indicates that SPF, DKIM, and DMARC authentication mechanisms successfully passed validation checks. However, these results do not imply legitimacy because the authentication applies only to the attacker-controlled domain `mnb-ny.com` rather than an official government domain. The email delivery path additionally shows routing through SendGrid's outbound infrastructure, indicating the use of a legitimate third-party email delivery service for improved delivery reliability and reduced suspicion. The combination of sender impersonation, trusted email services, and successful authentication records highlights an attempt to increase credibility and bypass conventional email security mechanisms.

MessageId	BgDe6LQOSC5UkphbQKcDsw@geopod-ismtpd-16				
Created at:	5/20/2026, 11:10:32 AM GMT+5:30 (Delivered after 2 sec)				
From:	 <ac.enforcementincometax.gov.in@mnb-ny.com>				
To:	adming@  .in				
Subject:	OFFICIAL COMPLIANCE NOTICE				
SPF:	pass with IP 149.72.120.130 Learn more				
DKIM:	pass with domain mnb-ny.com Learn more				
DMARC:	pass Learn more				

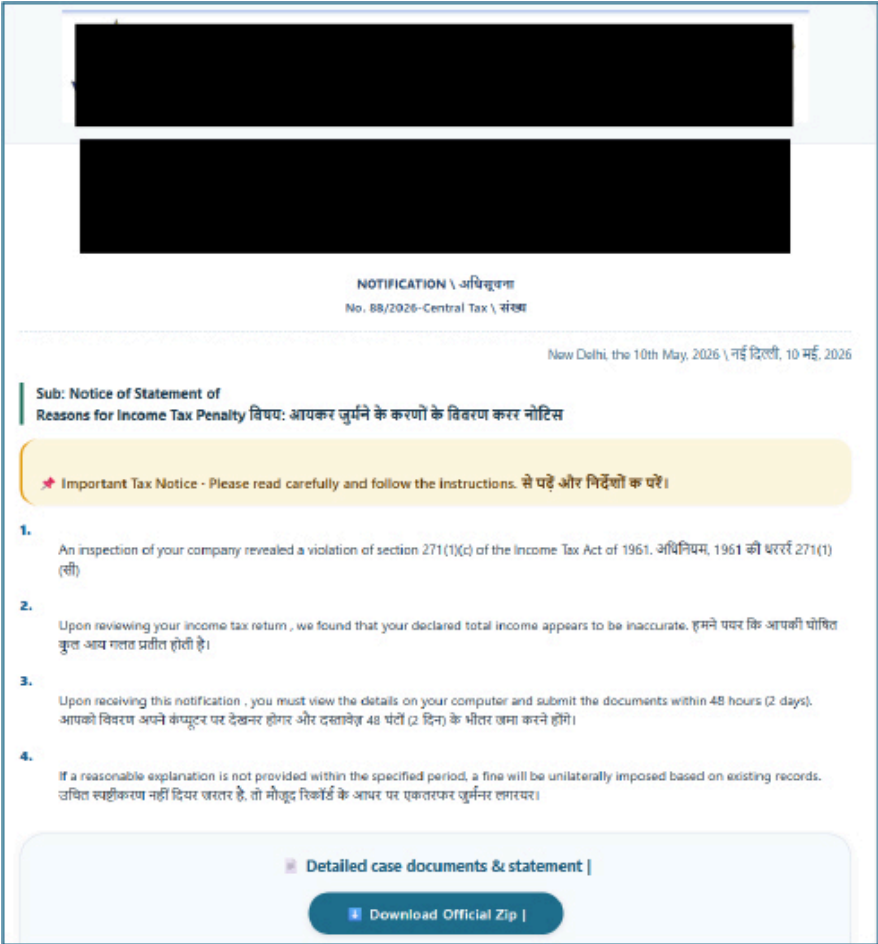
#	Delay	From *		To *	Protocol	Time received
0	2 sec	s.wrqvtrvn.outbound-mail.sendgrid.net.	→	[Google] mx.google.com	ESMTPS	5/20/2026, 11:10:34 AM GMT+5:30
1			→	[Google] 2002:a05:620a:4612:b0:912:bd42:b46f	SMTP	5/20/2026, 11:10:34 AM GMT+5:30
2			→	[Google] 2002:a5d:4405:0:b0:45a:b62b:4a62	SMTP	5/20/2026, 11:10:34 AM GMT+5:30

Email header analysis showing sender impersonation techniques, successful SPF/DKIM/DMARC validation, and the use of third-party email delivery infrastructure (SendGrid) to increase email credibility, delivery success rate, and social engineering effectiveness.

Phishing Website and Government Impersonation Analysis

The screenshot illustrates the malicious tax-themed phishing webpage designed to impersonate an official Indian government tax department and exploit user trust through government identity abuse. The page visually replicates legitimate government branding, including official logos, department names, and bilingual content (English and Hindi), to increase authenticity and reduce suspicion among victims. Such techniques are commonly used within social engineering campaigns to create psychological trust and encourage user interaction.

Further examination reveals several social engineering indicators embedded within the phishing page. The message contains urgency-driven language, tax penalty references, and time-sensitive compliance requests, specifically instructing victims to review details and submit documents within 48 hours. The webpage additionally includes a malicious download button disguised as an official document retrieval mechanism, ultimately leading users toward downloading the ZIP-based malware package. The combination of fear-based messaging, government impersonation, and deceptive download mechanisms indicates a deliberate attempt to manipulate victims into initiating the infection chain.



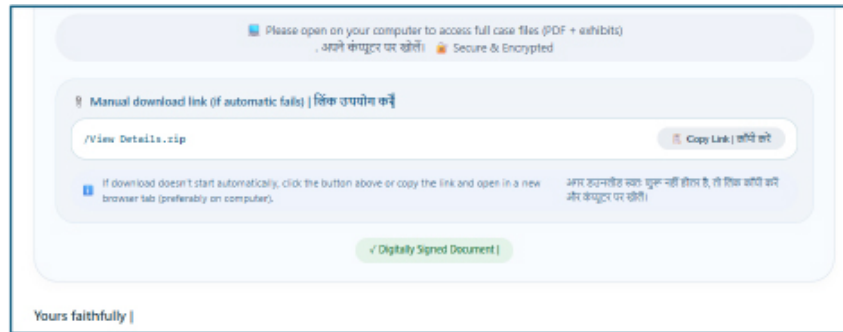


Figure 2: Tax-themed phishing webpage impersonating an Indian Tax department using government branding, bilingual content, urgency-based messaging, and a malicious download mechanism to facilitate social engineering-based malware delivery.

Multi-Region Government Impersonation and Infrastructure Reuse Analysis

The screenshot demonstrates that the same malicious infrastructure is not limited to impersonating Indian government entities but is also being reused to imitate an official Japanese government tax portal. The webpage attempts to visually replicate legitimate Japanese government branding, including language localization, navigation menus, official tax terminology, and user interface elements commonly found on authentic government platforms. This behavior indicates a broader multi-region phishing strategy designed to target victims across different geographical locations rather than a campaign focused on a single country.

A significant observation is that the same suspicious infrastructure previously used for Indian tax-themed phishing activity is now hosting a Japan-themed government impersonation portal, suggesting infrastructure reuse by the threat operators. Reusing infrastructure across multiple themes and regions is commonly observed in large-scale phishing operations, allowing threat actors to rapidly rotate lures while minimizing operational costs. The transition from Indian tax authority impersonation to Japanese tax authority impersonation indicates a flexible campaign architecture capable of dynamically changing themes depending on the intended target demographic. Such behavior increases the likelihood that the operators maintain a centralized phishing framework supporting multiple localized templates.

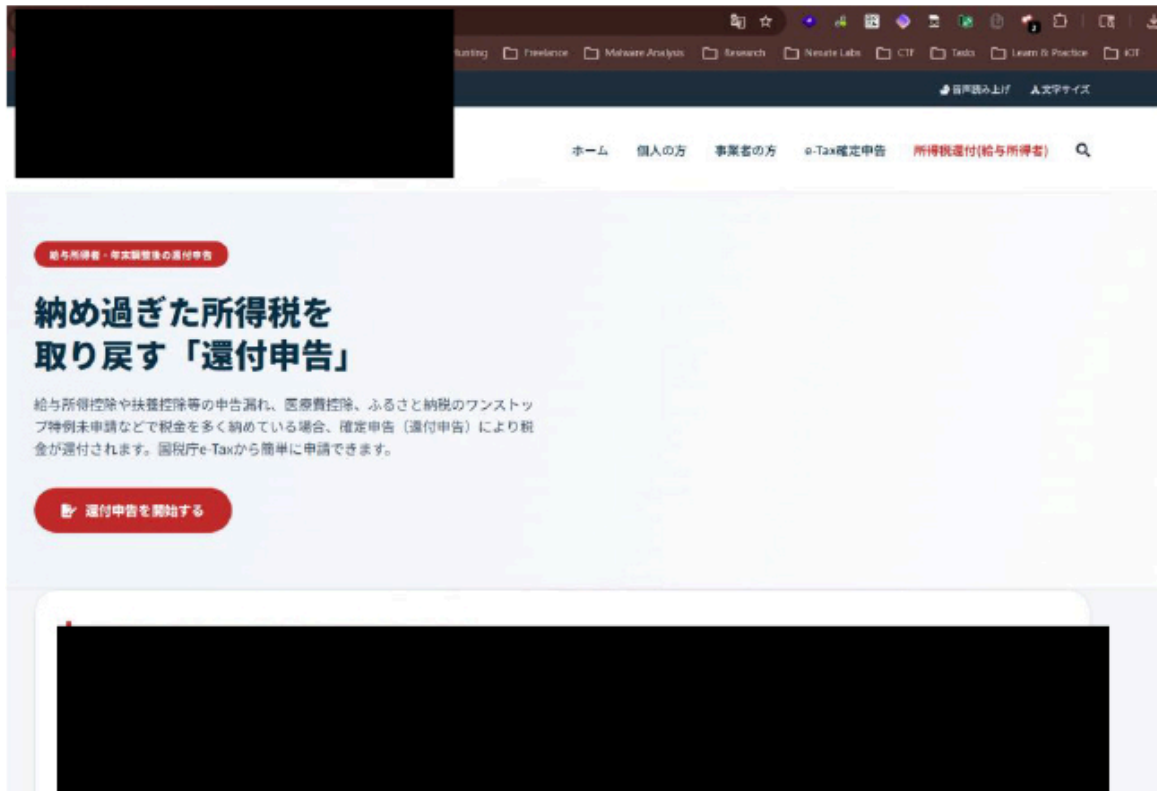


Figure 3: Japanese Government tax portal impersonation webpage hosted on previously identified suspicious infrastructure, demonstrating multi-region government impersonation, localized phishing templates, and infrastructure reuse techniques used to expand social engineering effectiveness across different target populations.

Chinese-Language Artifacts and Embedded Web Content Analysis

The screenshots reveal the presence of Chinese-language artifacts embedded within the phishing webpage source code and HTML metadata. The JavaScript snippet contains multiple Chinese comments associated with ZIP file processing and filename handling mechanisms, indicating that the phishing infrastructure or webpage components may have been developed using a Chinese-language development environment or reused from previously developed templates. The code references ZIP-related operations, including filename decoding and extraction functionality, which aligns with the campaign's delivery mechanism involving ZIP-based malware payload distribution.

The phrase “官方税务通知” translates to “Official Tax Notice” in English. The coexistence of English tax-related content with embedded Chinese-language strings is a notable operational artifact because legitimate Indian government portals would not normally contain such metadata. While these indicators may suggest Chinese-speaking operators, Chinese-language development tools, or template reuse, such artifacts alone are insufficient for definitive attribution. Therefore, these findings should be treated as supporting indicators rather than direct evidence of actor origin.

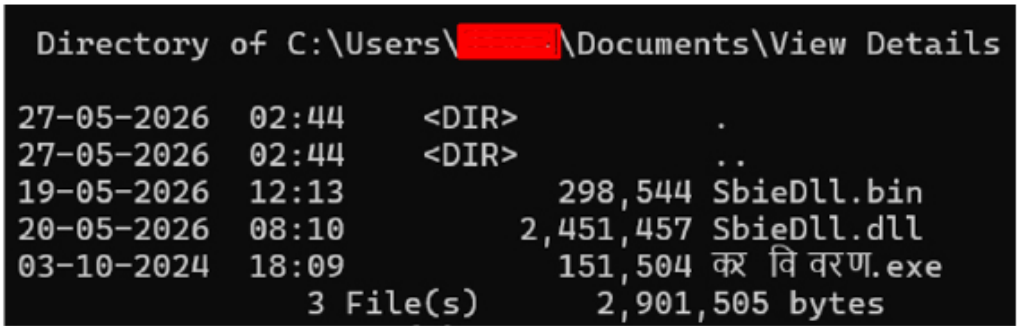
Figure 4: Embedded Chinese-language artifacts identified within JavaScript source code and HTML metadata, showing references to ZIP processing functionality and the page title “官方税务通知” (“Official

Tax Notice”), indicating potential template reuse, Chinese-language development environments, or Chinese-speaking operational indicators.

Payload Extraction and Staged Malware Component Analysis

The screenshot illustrates the contents extracted after downloading and unpacking the malicious ZIP archive, revealing a structured multi-stage malware package consisting of three primary components: कर विवरण.exe, SbieDll.dll, and SbieDll.bin. Rather than delivering a single executable payload, the attackers employ a modular payload architecture in which individual files perform dedicated roles during different stages of execution. This approach enables improved payload separation, execution control, and anti-analysis capabilities, making the infection chain more difficult to identify and disrupt.

Analysis of the extracted components indicates that कर विवरण.exe functions as the initial execution vector (Stage 0 Host Loader) responsible for environment preparation and malicious execution setup. The SbieDll.dll file acts as a polymorphic loader DLL, implementing API hooking, execution manipulation, and anti-analysis functionality, while SbieDll.bin serves as the encrypted shellcode payload that is later decrypted and executed in memory. The use of multiple staged components demonstrates an intentionally designed multi-stage infection mechanism, allowing threat actors to separate functionality and reduce direct exposure of the final malicious payload during initial execution.



```
Directory of C:\Users\... \Documents\View Details
27-05-2026  02:44    <DIR>          .
27-05-2026  02:44    <DIR>          ..
19-05-2026  12:13                298,544 SbieDll.bin
20-05-2026  08:10            2,451,457 SbieDll.dll
03-10-2024  18:09            151,504 कर विवरण.exe
                3 File(s)          2,901,505 bytes
```

Figure 5: Extracted contents of the malicious ZIP archive showing a multi-stage payload structure consisting of कर विवरण.exe (Stage 0 Host Loader), SbieDll.dll (Polymorphic Loader DLL), and SbieDll.bin (Encrypted Shellcode Payload) used to establish the complete malware execution chain.

Static Analysis of Initial Loader Executable (कर विवरण.exe)

The static analysis identified कर विवरण.exe as a 64-bit Portable Executable (PE64) compiled for AMD64 architecture using Microsoft Visual C/C++ with Visual Studio 2019. The sample contains debug information, overlay data, and certificate-related structures, indicating the presence of embedded resources or additional payload content. Although the filename attempts to appear legitimate through localized naming conventions, the executable primarily functions as the initial host loader responsible for environment preparation, dynamic API resolution, and transferring execution to subsequent malware stages.

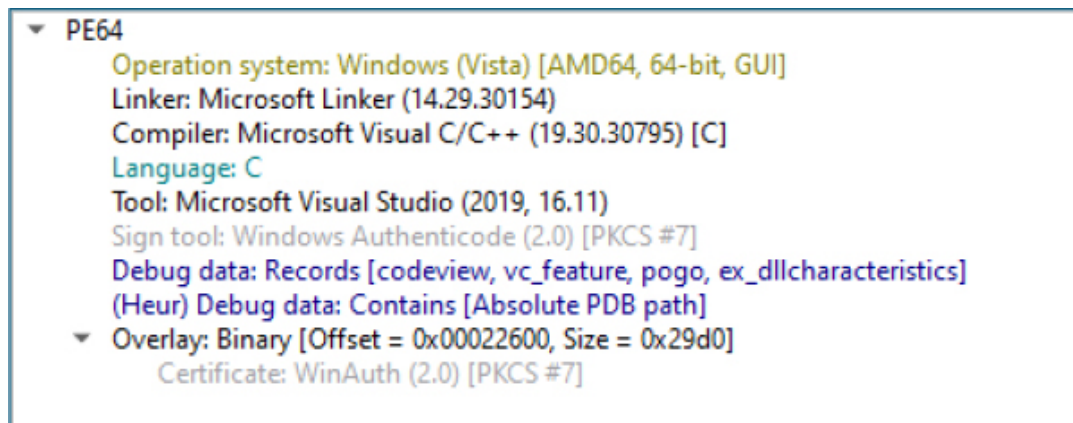


Figure 6: Static analysis results of कर विवरण.exe using Detect It Easy (DiE) showing PE64 architecture, Visual Studio compilation artifacts, debug information, and overlay structures, indicating its role as the initial execution vector responsible for preparing the multi-stage malware execution chain.

Static Analysis of Malicious Loader DLL (SbieDll.dll)

The static analysis identified SbieDll.dll as a 64-bit Portable Executable Dynamic Link Library (PE64 DLL) compiled for AMD64 architecture using Microsoft Visual C/C++ with Visual Studio 2022. The sample exhibited multiple anomalies commonly associated with packed or obfuscated malware, including modified DOS headers, abnormal section structures, repeated section names, and manipulated metadata timestamps. These indicators suggest the DLL was intentionally protected to hinder static analysis and reverse engineering efforts.

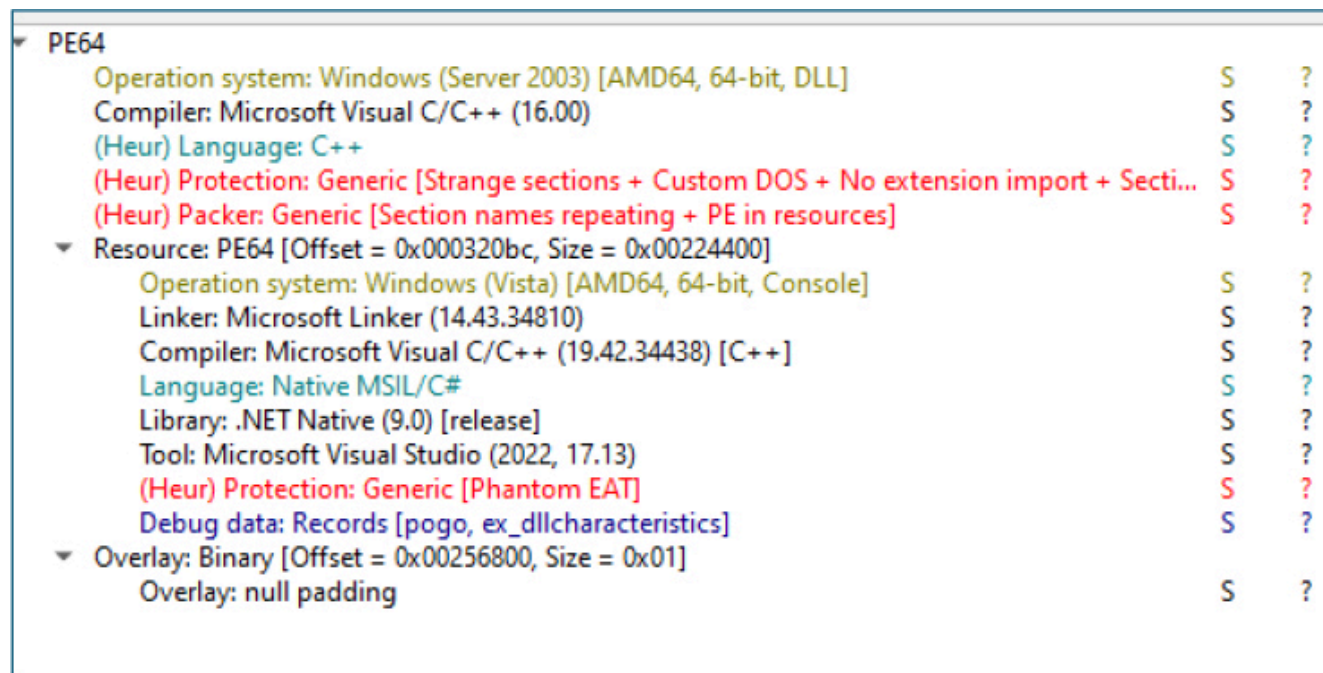


Figure 7: Static analysis results of SbieDll.dll using Detect It Easy (DiE) highlighting PE64 DLL architecture, packing and obfuscation indicators, timestamp anomalies, custom section structures, and anti-analysis characteristics, supporting its role as the polymorphic loader component within the multi-stage malware framework.

The extracted strings confirmed that the malware sample was obfuscated using the Astral-PE obfuscation tool.

Z%P%P%P%P%P%P%]%/
/CLR (WHITE) ...
/CLR (GRAY)Advanced utility for mutation (shallow obfuscation) of PE files....
GitHub: /CLR (BLUE) https://github.com/DosX-dev/Astral-PE/CLR (WHITE) 樟 ㊦
000102030405060708091011121314151617181920212223242526272829303132333435363738394041424344454647484950515253545556575859606162636465666768697071727374757677787980818283848586878889909192939495969798991001011021031041051061071081091101111121131141151161171181191201211221231241251261271281291301311321331341351361371381391401411421431441451461471481491501511521531541551561571581591601611621631641651661671681691701711721731741751761771781791801811821831841851861871881891901911921931941951961971981992002012022032042052062072082092102112122132142152162172182192202212222232242252262272282292302312322332342352362372382392402412422432442452462472482492502512522532542552562572582592602612622632642652662672682692702712722732742752762772782792802812822832842852862872882892902912922932942952962972982993003013023033043053063073083093103113123133143153163173183193203213223233234325326327328329330331332333334335336337338339340341342343344345346347348349350351352353354355356357358359360361362363364365366367368369370371372373374375376377378379380381382383384385386387388389390391392393394395396397398399400401402403404405406407408409410411412413414415416417418419420421422423424425426427428429430431432433434435436437438439440441442443444445446447448449450451452453454455456457458459460461462463464465466467468469470471472473474475476477478479480481482483484485486487488489490491492493494495496497498499500501502503504505506507508509510511512513514515516517518519520521522523524525526527528529530531532533534535536537538539540541542543544545546547548549550551552553554555556557558559560561562563564565566567568569570571572573574575576577578579580581582583584585586587588589590591592593594595596597598599600601602603604605606607608609610611612613614615616617618619620621622623624625626627628629630631632633634635636637638639640641642643644645646647648649650651652653654655656657658659660661662663664665666667668669670671672673674675676677678679680681682683684685686687688689690691692693694695696697698699700701702703704705706707708709710711712713714715716717718719720721722723724725726727728729730731732733734735736737738739740741742743744745746747748749750751752753754755756757758759760761762763764765766767768769770771772773774775776777778779780781782783784785786787788789790791792793794795796797798799800801802803804805806807808809810811812813814815816817818819820821822823824825826827828829830831832833834835836837838839840841842843844845846847848849850851852853854855856857858859860861862863864865866867868869870871872873874875876877878879880881882883884885886887888889890891892893894895896897898899900901902903904905906907908909910911912913914915916917918919920921922923924925926927928929930931932933934935936937938939940941942943944945946947948949950951952953954955956957958959960961962963964965966967968969970971972973974975976977978979980981982983984985986987988989990991992993994995996997998999100010011002100310041005100610071008100910101011101210131014101510161017101810191020102110221023102410251026102710281029103010311032103310341035103610371038103910401041104210431044104510461047104810491050105110521053105410551056105710581059106010611062106310641065106610671068106910701071107210731074107510761077107810791080108110821083108410851086108710881089109010911092109310941095109610971098109911001101110211031104110511061107110811091110111111121113111411151116111711181119112011211122112311241125112611271128112911301131113211331134113511361137113811391140114111421143114411451146114711481149115011511152115311541155115611571158115911601161116211631164116511661167116811691170117111721173117411751176117711781179118011811182118311841185118611871188118911901191119211931194119511961197119811991200120112021203120412051206120712081209121012111212121312141215121612171218121912201221122212231224122512261227122812291230123112321233123412351236123712381239124012411242124312441245124612471248124912501251125212531254125512561257125812591260126112621263126412651266126712681269127012711272127312741275127612771278127912801281128212831284128512861287128812891290129112921293129412951296129712981299130013011302130313041305130613071308130913101311131213131314131513161317131813191320132113221323132413251326132713281329133013311332133313341335133613371338133913401341134213431344134513461347134813491350135113521353135413551356135713581359136013611362136313641365136613671368136913701371137213731374137513761377137813791380138113821383138413851386138713881389139013911392139313941395139613971398139914001401140214031404140514061407140814091410141114121413141414151416141714181419142014211422142314241425142614271428142914301431143214331434143514361437143814391440144114421443144414451446144714481449145014511452145314541455145614571458145914601461146214631464146514661467146814691470147114721473147414751476147714781479148014811482148314841485148614871488148914901491149214931494149514961497149814991500150115021503150415051506150715081509151015111512151315141515151615171518151915201521152215231524152515261527152815291530153115321533153415351536153715381539154015411542154315441545154615471548154915501551155215531554155515561557155815591560156115621563156415651566156715681569157015711572157315741575157615771578157915801581158215831584158515861587158815891590159115921593159415951596159715981599160016011602160316041605160616071608160916101611161216131614161516161617161816191620162116221623162416251626162716281629163016311632163316341635163616371638163916401641164216431644164516461647164816491650165116521653165416551656165716581659166016611662166316641665166616671668166916701671167216731674167516761677167816791680168116821683168416851686168716881689169016911692169316941695169616971698169917001701170217031704170517061707170817091710171117121713171417151716171717181719172017211722172317241725172617271728172917301731173217331734173517361737173817391740174117421743174417451746174717481749175017511752175317541755175617571758175917601761176217631764176517661767176817691770177117721773177417751776177717781779178017811782178317841785178617871788178917901791179217931794179517961797179817991800180118021803180418051806180718081809181018111812181318141815181618171818181918201821182218231824182518261827182818291830183118321833183418351836183718381839184018411842184318441845184618471848184918501851185218531854185518561857185818591860186118621863186418651866186718681869187018711872187318741875187618771878187918801881188218831884188518861887188818891890189118921893189418951896189718981899190019011902190319041905190619071908190919101911191219131914191519161917191819191920192119221923192419251926192719281929193019311932193319341935193619371938193919401941194219431944194519461947194819491950195119521953195419551956195719581959196019611962196319641965196619671968196919701971197219731974197519761977197819791980198119821983198419851986198719881989199019911992199319941995199619971998199920002001200220032004200520062007200820092010201120122013201420152016201720182019202020212022202320242025202620272028202920302031203220332034203520362037203820392040204120422043204420452046204720482049205020512052205320542055205620572058205920602061206220632064206520662067206820692070207120722073207420752076207720782079208020812082208320842085208620872088208920902091209220932094209520962097209820992100210121022103210421052106210721082109211021112112211321142115211621172118211921202121212221232124212521262127212821292130213121322133213421352136213721382139214021412142214321442145214621472148214921502151215221532154215521562157215821592160216121622163216421652166216721682169217021712172217321742175217621772178217921802181218221832184218521862187218821892190219121922193219421952196219721982199220022012202220322042205220622072208220922102211221222132214221522162217221822192220222122222223222422252226222722282229223022312232223322342235223622372238223922402241224222432244224522462247224822492250225122522253225422552256225722582259226022612262226322642265226622672268226922702271227222732274227522762277227822792280228122822283228422852286228722882289229022912292229322942295229622972298229923002301230223032304230523062307230823092310231123122313231423152316231723182319232023212322232323242325232623272328232923302331233223332334233523362337233823392340234123422343234423452346234723482349235023512352235323542355235623572358235923602361236223632364236523662367236823692370237123722373237423752376237723782379238023812382238323842385238623872388238923902391239223932394239523962397239823992400240124022403240424052406240724082409241024112412241324142415241624172418241924202421242224232424242524262427242824292430243124322433243424352436243724382439244024412442244324442445244624472448244924502451245224532454245524562457245824592460246124622463246424652466246724682469247024712472247324742475247624772478247924802481248224832484248524862487248824892490249124922493249424952496249724982499250025012502250325042505250625072508250925102511251225132514251525162517251825192520252125222523252425252526252725282529253025312532253325342535253625372538253925402541254225432544254525462547254825492550255125522553255425552556255725582559256025612562256325642565256625672568256925702571257225732574257525762577257825792580258125822583258425852586258725882589259025912592259325942595259625972598259926002601260226032604260526062607260826092610261126122613261426152616261726182619262026212622262326242625262626272628262926302631263226332634263526362637263826392640264126422643264426452646264726482649265026512652265326542655265626572658265926602661266226632664266526662667266826692670267126722673267426752676267726782679268026812682268326842685268626872688268926902691269226932694269526962697269826992700270127022703270427052706270727082709271027112712271327142715271627172718271927202721272227232724272527262727272827292730273127322733273427352736273727382739274027412742274327442745274627472748274927502751275227532754275527562757275827592760276127622763276427652766276727682769277027712772277327742775277627772778277927802781278227832784278527862787278827892790279127922793279427952796279727982799280028012802280328042805280628072808280928102811281228132814281528162817281828192820282128222823282428252826282728282829283028312832283328342835283628372838283928402841284228432844284528462847284828492850285128522853285428552856285728582859286028612862286328642865286628672868286928702871287228732874287528762877287828792880288128822883288428852886288728882889289028912892289328942895289628972898289929002901290229032904290529062907290829092910291129122913291429152916291729182919292029212922292329242925292629272928292929302931293229332934293529362937293829392940294129422943294429452946294729482949295029512952295329542955295629572958295929602961296229632964296529662967296829692970297129722973297429752976297729782979298029812982298329842985298629872988298929902991299229932994299529962997299829993000300130023003300430053006300730083009301030113012301330143015301630173018301930203021302230233024302530263027302830293030303130323033303430353036303730383039304030413042304330443045304630473048304930503051305230533054305530563057305830593060306130623063306430653066306730683069307030713072307330743075307630773078307930803081308230833084308530863087308830893090309130923093309430953096309730983099310031013102310331043105310631073108310931103111311231133114311531163117311831193120312131223123312431253126312731283129313031313132313331343135313631373138313931403141314231433144314531463147314831493150315131523153315431553156315731583159316031613162316331643165316631673168316931703171317231733174317531763177317831793180318131823183318431853186318731883189319031913192319331943195319631973198319932003201320232033204320532063207320832093210321132123213321432153216321732183219322032213222322332234322532263227322832293230323132323233323432353236323732383239324032413242324332443245324632473248324932503251325232533254325532563257325832593260326132623263326432653266326732683269327032713272327332743275327632773278327932803281328232833284328532863287328832893290329132923293329432953296329732983299330033013302330333043305330633073308330933103311331233133314331533

```

MutexW = OpenMutexW(0x1F0001u, 0, L"SBIE_BOXED_ServiceCrypto_Mutex1");
v5 = 1;
if ( !MutexW && (MutexW = CreateMutexW(0, 1, L"SBIE_BOXED_ServiceCrypto_Mutex1")) == 0
    || WaitForSingleObject(MutexW, 0) )
{
    memset(&VersionInformation.dwMajorVersion, 0, 0x110u);
    VersionInformation.dwOSVersionInfoSize = 276;
    GetVersionExW(&VersionInformation);
    if ( VersionInformation.dwMajorVersion > 6
        || VersionInformation.dwMajorVersion == 6 && VersionInformation.dwMinorVersion )
    {
        hModule = LoadLibraryW(L"KernelBase.dll");
        qword_140021C08 = (__int64)LoadLibraryW(L"SecHost.dll");
        ModuleHandleW = GetModuleHandleW(L"ntdll.dll");
        NtCreateTimer2 = GetProcAddress(ModuleHandleW, "NtCreateTimer2");
        v9 = byte_140021C10;
        if ( NtCreateTimer2 )
            v9 = 1;
        byte_140021C10 = v9;
    }
    DuplicateHandle = ::DuplicateHandle;
    if ( hModule )
    {
        DuplicateHandle = (BOOL (__stdcall *)(HANDLE, HANDLE, HANDLE, LPHANDLE, DWORD, BOOL, DWORD))GetProcAddress(
                                                                    hModule,
                                                                    "DuplicateHandle");

        if ( !DuplicateHandle )
            DuplicateHandle = ::DuplicateHandle;
    }
    qword_140021C38 = (__int64 (__fastcall *)(_QWORD, _QWORD, _QWORD))SbieDll_Hook(
                                                                    (__int64)"DuplicateHandle",
                                                                    (__int64)DuplicateHandle,
                                                                    (__int64)sub_1400015E0);
}

```

WinMain Execution Logic and API Hook Registration

```

if ( hModule )
{
    GetTokenInformation = (BOOL (__stdcall *)(HANDLE, TOKEN_INFORMATION_CLASS, LPVOID, DWORD, PDWORD))GetProcAddress(hModule, "GetTokenInformation");
    if ( !GetTokenInformation )
        GetTokenInformation = ::GetTokenInformation;
}
qword_140021C30 = SbieDll_Hook((__int64)"GetTokenInformation", (__int64)GetTokenInformation, (__int64)sub_140001680);
if ( qword_140021C30 && v16 )
{
    dwTlsIndex = TlsAlloc();
    wcsncpy(Value, L"CryptSvc");
    v6 = sub_1400018F0(Value);
    v19 = (_QWORD *)v6;
    if ( !v6 )
        return v6;
    qword_140021B80 = Scm_HookSetServiceStatus(sub_1400016E0);
    *v19 = 0;
    LibraryW = LoadLibraryW(L"cryptsvc.dll");
    if ( LibraryW )
    {
        CryptServiceMain = GetProcAddress(LibraryW, "CryptServiceMain");
        if ( CryptServiceMain )
        {
            ProcessHeap = GetProcessHeap();
            v25 = (SERVICE_TABLE_ENTRYW *)HeapAlloc(ProcessHeap, 4u, 0x200u);
            v25[1].lpServiceName = 0;
            v25[1].lpServiceProc = 0;
            v26 = lpName;
            v25->lpServiceName = Value;
            v25->lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)CryptServiceMain;
            SetEnvironmentVariableW(v26, Value);
            StartServiceCtrlDispatcherW(v25);
            v5 = 0;
        }
    }
}

```

Service Initialization and CryptSvc Registration Logic

Hook 1 – AccessCheckByType Privilege Bypass Mechanism

The screenshots demonstrate the implementation of Hook 1, targeting the Windows `AccessCheckByType()` API responsible for validating whether a process or thread has sufficient permissions to access protected resources. The malware registers a malicious replacement function through `SbieDll_Hook()`, redirecting the legitimate API execution to a custom handler identified as `sub_1400014D0`. Analysis of the replacement routine shows that it directly assigns `GrantedAccess = 0xFFFFFFFF` (full access permissions) and sets `AccessStatus = TRUE`, forcing the operating system to treat all access requests as successful regardless of actual security restrictions.

By bypassing normal Windows access control mechanisms, every authorization check performed within the hooked process automatically succeeds. This effectively removes privilege boundaries and allows the malware to access protected objects, open restricted handles, manipulate process tokens, and interact with sensitive system resources without encountering permission limitations. Such behavior significantly increases malware capabilities and enables unrestricted interaction with otherwise protected system components.

```
sub_1400014D0 proc near
arg_48= qword ptr 50h
arg_50= qword ptr 58h

sub     rsp, 28h
mov     rcx, [rsp+28h+arg_50]
mov     rax, [rsp+28h+arg_48]
mov     dword ptr [rax], 0FFFFFFFFh
mov     dword ptr [rcx], 1
xor     ecx, ecx        ; dwErrCode
call    cs:SetLastError
mov     eax, 1
add     rsp, 28h
retn
sub_1400014D0 endp
```

AccessCheckByType Hook Registration through SbieDll_Hook()

```
v13 = SbieDll_Hook((__int64)"AccessCheckByType", (__int64)AccessCheckByType, (__int64)sub_1400014D0);
v14 = v13 == 0;
qword_140021C20 = v13;
SetThreadToken = ::SetThreadToken;
v16 = !v14;
if ( hModule )
{
    SetThreadToken = (BOOL (__stdcall *) (PHANDLE, HANDLE))GetProcAddress(hModule, "SetThreadToken");
    if ( !SetThreadToken )
        SetThreadToken = ::SetThreadToken;
}
```

Malicious replacement routine forcing successful access validation

Hook 2 – CreateFileW Path Rewriting Mechanism

The implementation of Hook 2 targets the Windows CreateFileW() API responsible for creating and opening files, devices, and system resources. The malware intercepts file access requests and performs path manipulation before passing execution to the original function. Analysis of the code reveals logic that checks for specific paths, including \\.\PhysicalDrive and \system32\CatRoot2\, indicating that the malware selectively modifies how file operations are handled for system resources.

By intercepting and modifying file-related parameters, the malware gains the ability to alter normal file access behaviour and redirect execution toward controlled paths or resources. Such behaviour can be used to evade monitoring mechanisms, hide malicious activity, manipulate access to protected system files, or ensure subsequent malware stages interact with attacker-controlled resources rather than legitimate locations. The use of CreateFileW API manipulation provides an additional layer of execution control and defense evasion within the overall malware lifecycle.

```
int64 __fastcall sub_140001500(wchar_t *Str2, unsigned int a2, unsigned int a3, int64 a4)
{
    wchar_t *v8; // rsi
    WCHAR Buffer[328]; // [rsp+40h] [rbp-298h] BYREF

    if ( !wcsnicmp(Str2, L"\\\\.\\PhysicalDrive", 0x11u) && !wcschr(Str2 + 17, 0x5Cu) && a2 == 0x80000000 )
        a2 = 0;
    v8 = wcsstr(Str2, L"\\system32\\CatRoot2\\");
    if ( v8 )
    {
        GetWindowsDirectoryW(Buffer, 0x104u);
        if ( !wcsnicmp(Buffer, Str2, v8 - Str2) && a2 == 0x80000000 )
            a2 = -1073741824;
    }
    return qword_140021C40(Str2, a2, a3, a4);
}
```

CreateFileW path manipulation and conditional file handling logic

Hook 3 – DuplicateHandle Error Suppression Mechanism

The implementation of Hook 3 targets the Windows DuplicateHandle() API responsible for duplicating object handles across processes and threads. The malware intercepts the original API execution and monitors for specific failure conditions, particularly ERROR_ACCESS_DENIED (Error Code: 5). Instead of allowing the operation to fail normally, the replacement routine generates an alternative execution path by creating a new object through CreateEventW(), clearing the error state using SetLastError(0), and forcing a successful return value.

This behaviour effectively acts as an error suppression and execution recovery mechanism, preventing access-denied failures from interrupting malware execution. By masking permission-related issues and returning artificially successful results, the malware ensures that subsequent operations continue without interruption, increasing execution reliability and reducing the likelihood of operational failure caused by security restrictions or access limitations. Such techniques are commonly used to maintain stable execution flow and conceal abnormal behaviour from monitoring mechanisms.

```

__int64 __fastcall sub_1400015E0(__int64 a1, __int64 a2, __int64 a3, HANDLE *a4)
{
    __int64 result; // rax

    result = qword_140021C38(a1, a2, a3);
    if ( !(_DWORD)result )
    {
        if ( GetLastError() == 5 && a1 != -1 && a3 == -1 )
        {
            *a4 = CreateEventW(0, 0, 0, 0);
            SetLastError(0);
            return 1;
        }
        else
        {
            return 0;
        }
    }
    return result;
}

```

DuplicateHandle error interception and failure suppression logic

Hook 4 SetThreadToken Impersonation and Token Manipulation Mechanism

The implementation of Hook 4 targets the Windows SetThreadToken() API responsible for assigning security tokens to execution threads. The hook monitors for scenarios where SetThreadToken() fails with ERROR_ACCESS_DENIED (Error Code: 5) while operating on the current execution thread. Instead of allowing execution failure, the malware invokes OpenProcessToken() using TOKEN_ALL_ACCESS (0xF01FF) permissions to obtain the current process token and subsequently duplicates it using DuplicateToken() with SecurityImpersonation privileges. The duplicated token is then associated with the current thread, allowing the malware to continue execution under an impersonated security context.

This mechanism enables the malware to bypass normal security restrictions and manipulate thread-level privileges without interrupting execution flow. Additionally, the original token information is stored within Thread Local Storage (TLS) for later retrieval and manipulation by subsequent hooks. The use of token impersonation techniques significantly enhances malware capabilities by allowing execution within elevated contexts while reducing failures caused by permission restrictions.

```

{
    __int64 result; // rax
    unsigned int v5; // ebx
    DWORD v6; // ecx
    void *TokenHandle; // [rsp+40h] [rbp+18h] BYREF
    void *DuplicateTokenHandle; // [rsp+48h] [rbp+20h] BYREF

    TlsSetValue(dwTlsIndex, 0);
    result = qword_140021C28(a1, a2);
    v5 = result;
    if ( !(_DWORD)result )
    {
        if ( GetLastError() == 5 && !a1 )
        {
            v5 = OpenProcessToken((HANDLE)0xFFFFFFFFFFFFFFFF, 0xF01FFu, &TokenHandle);
            if ( v5
                && (v5 = DuplicateToken(TokenHandle, SecurityImpersonation, &DuplicateTokenHandle), CloseHandle(TokenHandle), v5)
                && (v5 = qword_140021C28(0, DuplicateTokenHandle), CloseHandle(DuplicateTokenHandle), v5) )
            {
                TlsSetValue(dwTlsIndex, a2);
                v6 = 0;
            }
            else
            {
                v6 = 5;
            }
            SetLastError(v6);
        }
        return v5;
    }
    return result;
}

```

SetThreadToken interception and token duplication logic

Hook 5 – GetTokenInformation TLS-Based Token Substitution

The implementation of Hook 5 targets the Windows GetTokenInformation() API responsible for retrieving security token attributes and permission-related information. Analysis of the code shows that the malware retrieves previously stored token information from Thread Local Storage (TLS) using TlsGetValue(dwTlsIndex) and conditionally replaces the original token reference before forwarding execution to the legitimate function.

By intercepting token retrieval operations and substituting stored values, the malware can manipulate how token-related information appears during execution. This behaviour assists in concealing prior token manipulation activities, maintaining execution consistency, and ensuring that subsequent components interact with attacker-controlled token contexts rather than original security information. The use of TLS-based token substitution creates an additional abstraction layer that complicates behavioural analysis and strengthens the malware's overall defense-evasion capabilities.

```

LPVOID Value; // rax

Value = TlsGetValue(dwTlsIndex);
if ( Value )
    a1 = Value;
return qword_140021C30(a1, a2, a3, a4);

```

GetTokenInformation TLS retrieval and token substitution logic.

Stage 1 Analysis — Polymorphic Loader DLL

Mersenne Twister VM Dispatcher

The execution logic of Stage 1 is implemented within the malicious Sbiedll.dll component loaded through DLL Search Order Hijacking. Windows normally searches the application's local directory before system directories when resolving DLL dependencies, allowing the malware to force execution of the malicious Sbiedll.dll instead of the legitimate Sandboxie library. This component functions as the campaign's primary anti-analysis and execution control layer, responsible for initiating complex runtime behaviour before transitioning to subsequent payload stages.

Further analysis of the code reveals the implementation of a 64-bit Mersenne Twister (MT19937-64) based execution dispatcher, which initializes its internal state using GetTickCount64() as a dynamic seed source. The observed mathematical operations and repeated state-generation logic indicate the use of a pseudo-random execution engine capable of generating varying execution patterns during runtime. This mechanism likely serves as a custom Virtual Machine (VM) dispatcher, ensuring that execution characteristics differ across infections and making behavioural analysis, signature generation, and execution tracing significantly more difficult. Such polymorphic behaviour reduces consistency between executions and strengthens the malware's overall defense-evasion capabilities.

```
TickCount64 = (unsigned int)GetTickCount64();
*((_QWORD *)&v12[318] + 1) = 0x5555555555555555LL;
*((_QWORD *)&v12[6] + 1) = TickCount64;
for ( i = 1; i < 0x138; ++i )
{
    TickCount64 = i + 0x5851F42D4C957F2DLL * ((TickCount64 >> 62) ^ TickCount64);
    *((_QWORD *)&v12[6] + i + 1) = TickCount64;
}
LODWORD(v12[6]) = 312;
v11 = 0;
if ( (unsigned __int8)sub_18000AAA0(i, TickCount64, 0x5851F42D4C957F2DLL) )
{
    v9 = &std::_Func_impl_no_alloc<_lambda_2785c2a111f36350acb5c0e181421178_,void,>::_vftable';
    v10 = *a1;
    v11 = &v9;
}
```

Mersenne Twister VM initialization and state generation logic

MT19937-64 Core State Engine Analysis

The screenshots demonstrate the implementation of the MT19937-64 Core State Engine embedded within the malicious Sbiedll.dll component. Analysis of the code reveals mathematical operations and state-generation routines characteristic of the 64-bit Mersenne Twister pseudo-random number generator (PRNG). The observed logic initializes and continuously updates an internal 624-element state array, where each generated state value is derived from previous entries through a series of bitwise XOR operations, right-shift calculations, conditional masking, and predefined constants, including

0xB5026F5AA96619E9. These operations align with the standard MT19937-64 state transition algorithm, which is widely known for generating statistically random outputs across large periods.

Within the malware execution chain, this MT19937-64 implementation appears to function as a dynamic execution engine responsible for introducing runtime variability and polymorphic behaviour. Rather than producing predictable execution paths, the generated pseudo-random values can influence internal execution states, dispatch routines, or subsequent transformations used by the malware. This behaviour significantly complicates reverse engineering, behavioural analysis, and signature-based detection, since execution characteristics may differ across environments and individual infections. The integration of a custom PRNG-driven execution model demonstrates deliberate anti-analysis and evasion-oriented design choices within the malware framework.

```
do
{
    v6 = v5[1];
    ++v5;
    v7 = v6 ^ (v4 ^ v6) & 0xFFFFFFFF80000000uLL;
    v4 = v6;
    v5[311] = v5[155] ^ (v7 >> 1) ^ -((__int64)((v6 & 1) != 0) & 0xB5026F5AA96619E9uLL;
    v2;
}
```

MT19937-64 internal state generation logic

```
do
{
    v11 = *(_QWORD*)(v8 + 8);
    v8 += 8;
    v12 = v11 ^ (v10 ^ v11) & 0xFFFFFFFF80000000uLL;
    v10 = v11;
    *(_QWORD*)(v8 - 2504) = *(_QWORD*)(v8 + 1240) ^ (v12 >> 1) ^ -((__int64)((v11 & 1) != 0) & 0xB5026F5AA96619E9uLL;
    --v9;
}
```

MT19937-64 state transformation operations

```
v18 = *(_QWORD*)(a1 + 8LL * v1 + 8);
*(_DWORD*)a1 = v1 + 1;
v19 = v18 ^ *(_QWORD*)(a1 + 5000) & (v18 >> 29);
v20 = v19
    ^ ((v19 & 0x38EB3FFFF6D3LL) << 17)
    ^ (((v19 ^ ((v19 & 0x38EB3FFFF6D3LL) << 17)) & 0xFFFFFFFFFFFFBF77uLL) << 37);
return v20 ^ (v20 >> 43);
```

Pseudo-random value generation and output calculation

Non-Linear TLS-Based String Decryption

The screenshot demonstrates the implementation of a Thread Local Storage (TLS)-based string transformation and decryption routine embedded within SbieDll.dll. Analysis of the code reveals

operations involving `NtCurrentTeb()`, `ThreadLocalStoragePointer`, and `TlsIndex`, indicating that the malware stores and retrieves data through thread-specific storage locations rather than maintaining static values within global memory structures. The routine performs multiple mathematical transformations, including XOR operations, offset calculations, and dynamic memory access, before reconstructing internal values during runtime.

Unlike conventional decryption routines that rely on static keys and linear processing sequences, this implementation appears to utilize a non-linear transformation approach, where reconstructed values depend on TLS-derived context information and dynamically calculated operations. Such behaviour makes the extraction of embedded strings significantly more difficult because the decrypted content may vary depending on execution state and thread context. This technique strengthens the malware's anti-analysis capabilities by hiding important artifacts such as API names, configuration values, and internal execution strings until runtime execution occurs.

```
{
    if ( i % 2 )
        v2 = 163;
    else
        v2 = 87;
    v3 = (((int)*((unsigned __int16 *)qword_180012800 + (i + 2) % 8) >> 4)
        | (16 * *((unsigned __int16 *)qword_180012800 + (i + 1) % 8)))
        ^ *((unsigned __int16 *)qword_180012800 + i % 8);
    *(_WORD *)*((_QWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + (unsigned int)TlsIndex + 35424LL + 2 * i) = v2 ^ v3 ^ *(_WORD *)sub_18000A310(qword_1800300F8, i);
}
*(_WORD *)*((_QWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + (unsigned int)TlsIndex + 35448LL) = 0;
return *((_QWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + (unsigned int)TlsIndex + 35424LL);
}
```

TLS-based string transformation and decryption logic

COM IContextCallback Execution Handler and Thread Evasion Mechanism

The screenshot demonstrates the implementation of a COM-based execution mechanism used by the malware to execute payload components while avoiding traditional thread creation APIs. Analysis of the code shows references to `CoInitializeEx()`, `CoGetObjectContext()`, and COM-related callback handling routines, indicating that the malware leverages COM object infrastructure instead of directly invoking APIs such as `CreateThread()`. The loader dynamically resolves these functions and registers execution handlers through COM interfaces before transferring execution to the next payload stage.

Rather than creating a new thread through standard Windows APIs that are frequently monitored by Endpoint Detection and Response (EDR) solutions, the malware appears to execute shellcode through a `COM IContextCallback::ContextCallback` handler, causing execution to occur within legitimate COM apartment thread contexts. This technique reduces behavioural visibility because the shellcode execution appears to originate from normal COM activity rather than suspicious thread creation events. By blending malicious activity with legitimate system behaviour, the malware significantly strengthens its defense-evasion and anti-analysis capabilities.

```

v29 = (__int64 (__fastcall **)(__QWORD))sub_1800046B0(v64, memcpy);
v50 = (void (__fastcall *)(__int64, __int64, _QWORD))(**v29)(v29);
v49 = v70;
v50(v9, v6 + 4, v70);
v10 = (__int64 (__fastcall **)(__QWORD))sub_1800038D0(v65, CoInitializeEx);
v51 = (__int64 (__fastcall *)(__QWORD, _QWORD))(**v10)(v10);
v5 = v51(0, 0);
v11 = (__int64 (__fastcall **)(__QWORD))sub_180003680(v66, CoGetObjectContext);
v30 = (__int64 (__fastcall *)(_BYTE *, __int64 *))(**v11)(v11);
v5 = v30(v71, &v68);
v31 = *(__int64 (__fastcall **)(__int64, __int64, __int64, _BYTE *, _DWORD))(*(__QWORD *)v68 + 64LL);
v5 = v31(v68, v9, v6, v71, 0);
v12 = (__int64 (__fastcall **)(__QWORD))sub_180003430(v67, Sleep);
v32 = (void (__fastcall *)(__int64))(**v12)(v12);
v32(0xFFFFFFFF);

```

COM callback registration and execution handling logic

Stage 2: Encrypted Shellcode & Reflective PE Loader (SbieDll.bin)

Dynamic API Hashing via ROR13 PEB Walk

The screenshots demonstrate functionality from Stage 2, where the malicious SbieDll.bin component performs dynamic API resolution using a Process Environment Block (PEB) walk combined with ROR13 hashing operations. The code begins by accessing `__readgsqword(0x60)`, which retrieves a pointer to the Process Environment Block (PEB) in a 64-bit environment. Rather than relying on static imports that can be easily identified during analysis, the malware traverses internal process structures to enumerate loaded modules and resolve required APIs dynamically during runtime.

Further analysis reveals repeated ROR4/ROR13 rotation operations, character processing, and hash calculations used to generate API identifiers instead of storing readable function names. The malware compares these calculated hash values against internally stored values to locate target APIs and obtain their addresses during execution. This technique hides important APIs from static analysis tools and significantly complicates reverse engineering efforts because analysts cannot directly observe imported functions through conventional import tables. The use of dynamic API hashing enhances the malware's anti-analysis, obfuscation, and defense-evasion capabilities, while reducing dependency on traditional executable structures.

PEB access through `__readgsqword()` for module enumeration

```

while ( 1 )
{
    v16 = (char *) (v7 + *v15);
    v17 = 0;
    while ( 1 )
    {
        v20 = *v16;
        if ( !*v16 )
            break;
        v18 = __ROR4__(v17, 13);
        v19 = v20 - 32;
        if ( v20 < 97 )
            v19 = v20;
        v17 = v19 + v18;
        ++v16;
    }
    if ( a4 == v17 + v10 )
        return v7
        + *(unsigned int *) (v7
        + *(unsigned int *) ((unsigned int) v8 + v7 + 28)
        + 4LL * *(unsigned __int16 *) (v7 + *(unsigned int *) ((unsigned int) v8 + v7 + 36) + 2 * v14));
    v14 = (unsigned int) (v14 + 1);
    ++v15;
    if ( (unsigned int) v14 >= *(_DWORD *) ((unsigned int) v8 + v7 + 24) )
        goto LABEL_17;
}

```

ROR13 hashing routine and dynamic API resolution logic

Mutated RC4 Stream Cipher

The screenshots demonstrate the implementation of a modified RC4-based stream cipher used within Stage 2 (SbieDll.bin) for payload decryption and protection of embedded shellcode components. The first code section resembles the traditional RC4 Key Scheduling Algorithm (KSA), where an initial 256-byte state array is created and rearranged using key-dependent operations. However, additional variables and modified state manipulations indicate that the malware does not implement a standard RC4 algorithm directly and instead introduces custom transformations to alter the internal state generation process.

The second code segment demonstrates functionality like the Pseudo-Random Generation Algorithm (PRGA) used in RC4, where dynamic state values are continuously updated and combined with payload data through XOR-based operations. Unlike conventional RC4 implementations, the observed modifications introduce additional calculations and state transitions that alter the generated keystream. This mutated RC4 implementation likely serves as an additional anti-analysis mechanism, making payload decryption more difficult and reducing the effectiveness of automated detection tools that rely on identifying traditional RC4 patterns. The decrypted output is subsequently used to reconstruct and execute the next memory-resident payload stage.

```

for ( i = 0; i < 256; ++i )
    *(_BYTE *) (a4 + i) = i;
for ( j = 0; j < 256; ++j )
{
    v6 = (*(_unsigned __int8 *) (a3 + j % a5) + *(_unsigned __int8 *) (a4 + j) + v6) % 256;
    v7 = *(_BYTE *) (a4 + j);
    *(_BYTE *) (a4 + j) = *(_BYTE *) (a4 + v6);
    *(_BYTE *) (a4 + v6) = v7;
}
*(_BYTE *) (a4 + 256) = 0;
result = a4;
*(_BYTE *) (a4 + 257) = 0;
return result;

```

Modified RC4 Key Scheduling Algorithm (KSA) implementation

```
for ( i = 0; ; ++i )
{
    result = a5;
    if ( i >= a5 )
        break;
    *(_BYTE *)(a4 + 257) += *(_BYTE *)(a4 + (unsigned __int8)++*(_BYTE *)(a4 + 256));
    v6 = *(_BYTE *)(a4 + *(_BYTE *)(a4 + 256));
    *(_BYTE *)(a4 + *(_BYTE *)(a4 + 256)) = *(_BYTE *)(a4 + *(_BYTE *)(a4 + 257));
    *(_BYTE *)(a4 + *(_BYTE *)(a4 + 257)) = v6;
    *(_BYTE *)(i + a3) ^= *(_BYTE *)(a4 + *(_BYTE *)(a4 + 257))
        ^ *(_BYTE *)(a4 + *(_BYTE *)(a4 + 256));
}
```

Modified RC4 Pseudo-Random Generation Algorithm (PRGA) logic

Reflective PE Loader Engine and In-Memory Payload Execution

The screenshots demonstrate the implementation of a Reflective PE Loader Engine responsible for loading the decrypted payload entirely within memory without relying on the standard Windows loading process. Analysis of the code reveals references to memory-related APIs, including `ZwFreeVirtualMemory` and `ZwAllocateVirtualMemory`, indicating direct manipulation of process memory regions during runtime. The malware allocates memory dynamically, reconstructs required executable structures, and prepares the decrypted payload for execution without invoking conventional loading mechanisms such as `LoadLibrary()`. This behaviour indicates that the malware performs manual Portable Executable (PE) loading rather than allowing the operating system loader to manage execution.

Further analysis of the memory allocation and execution logic suggests that the malware manually handles critical PE loading operations, including memory allocation, section mapping, address resolution, and execution preparation, entirely in user space. By avoiding disk writes, standard library loading routines, and native loader involvement, the malware significantly reduces forensic artifacts and bypasses security mechanisms that commonly monitor conventional process loading behaviour. This technique represents a classic implementation of Reflective PE Injection, enabling the final payload to remain memory-resident while improving stealth and strengthening defense-evasion capabilities.

```
strcpy(v46, "ZwFreeVirtualMemory");
v11 = sub_20C(a1);
v13 = (__int64 (__fastcall *))(__int64, __int64, char *, __int64)sub_100(a1, -1997096815, v12, -1997096815);
v14 = (void (__fastcall *)(_QWORD, __int64, __int64 *, __int64 *, __int64 *, __int64))v13(a1, 2297870481LL, v46, v11);
strcpy(v47, "ZwAllocateVirtualMemory");
v15 = sub_20C(v14);
v17 = (__int64 (__fastcall *))(void (__fastcall *)(_QWORD, __int64, __int64 *, __int64 *, __int64 *, __int64, char *, __int64))sub_100((__QWORD)v14, -1997096815, v16, -1997096815);
v18 = (int (__fastcall *)(__int64, __int64, __int64 *, __int64, __QWORD, __int64 *))v17(v14, 2297870481LL, v47, v15);
```

Figure 26: Memory allocation and payload preparation routines

```

v19 = a4 + *(int *)(a4 + 60);
v20 = v18;
a8 = *(unsigned int *)(v19 + 80);
a9 = *(_QWORD *)(v19 + 48);
if ( v18((__int64)v14, 4, &a9, -1, 0, &a8) < 0 )
{
    a9 = 0;
    if ( v20((__int64)v14, 4, &a9, -1, 0, &a8) < 0 )
        return 0;
}

```

Reflective PE loading and execution logic

Control Flow Flattening (CFF) Obfuscation

The screenshot demonstrates the implementation of Control Flow Flattening (CFF) used to protect the malware's core execution routines from static analysis and reverse engineering. Analysis of the code reveals multiple mathematical transformation operations, including `__ROR4__`, `__ROL4__`, and `_byteswap_ulong`, which are applied to internal state values before transferring execution through an indirect `jmp rax` instruction. Instead of maintaining a traditional linear execution flow, where functions follow predictable branches and control paths, the malware restructures execution into dynamically computed transitions controlled by runtime state variables.

This behaviour is consistent with LLVM-based Control Flow Flattening, where multiple execution branches are collapsed into a centralized dispatcher mechanism that computes the next execution target dynamically. As a result, the original logical structure becomes hidden behind multiple mathematical transformations and indirect jumps, making execution tracing significantly more difficult. Such techniques disrupt IDA Pro graph reconstruction, reduce the effectiveness of signature-based detection, and complicate behavioural analysis, as analysts cannot easily identify the original execution sequence. The use of CFF obfuscation demonstrates a deliberate anti-analysis strategy intended to increase reverse engineering complexity and conceal critical malware functionality.

```

if ( !(*(unsigned int (__fastcall **)(__int64, __int64, __int64, __int64))&unk_43440
- __ROR4__(
    _byteswap_ulong(
        __ROL4__(
            -__ROR4__(-__ROR4__(dword_43590[dword_43638], dword_43638 - 38), dword_43638 - 38),
            -38 - dword_43638)),
        dword_43638 - 38))) (
    a1,
    a2,
    112,
    a4) )
asm { jmp rax }

```

Control Flow Flattening dispatcher and state transformation logic

WebSocket-Based Command-and-Control (C2) Tunneling via libwebsockets

The screenshots demonstrate the implementation of WebSocket-based Command-and-Control (C2) communication through the libwebsockets library within the malware framework. Analysis of the identified symbols, including `lws_http_serve`, `lws_http_to_callback`, and `lws_http_transaction_completed`, confirms the presence of libwebsockets-related functionality, indicating that the malware leverages legitimate networking libraries to establish covert communication channels. Rather than initiating direct or easily identifiable C2 traffic, the malware begins communication using a standard HTTP GET request, after which the connection is upgraded through HTTP/1.1 101 Switching Protocols, identical to the mechanism used by legitimate web applications when establishing WebSocket sessions.

This communication technique allows malicious traffic to blend seamlessly with normal web activity, making network-level detection significantly more difficult. Additional artifacts such as `CONNECT %s:%u HTTP/1.0` indicate that the malware is proxy-aware, allowing C2 traffic to be tunnelled through enterprise HTTP proxy environments to bypass network restrictions and filtering mechanisms. The absence of visible hardcoded IP addresses or domain references further suggests that C2 configuration is dynamically supplied during earlier execution stages, preventing straightforward extraction of infrastructure indicators through static analysis. This combination of WebSocket tunneling, proxy support, and dynamic configuration retrieval significantly enhances the malware's stealth, resilience, and defense-evasion capabilities.

	seg000:000000...	0000000F	C	<code>lws_http_serve</code>
	seg000:000000...	00000015	C	<code>lws_http_to_fallback</code>
	seg000:000000...	0000001F	C	<code>lws_http_transaction_completed</code>

libwebsockets-related function references

HTTP/1.1 WebSocket upgrade response artifacts

Dynamic Behaviour:

Upon execution, the malware loaded `SbieDll.dll` and performed process injection into multiple running processes, including `svchost.exe` and `mitmweb.exe`. The DLL exhibited broad injection behaviour by dynamically attaching itself to newly launched processes, indicating the presence of system-wide API hooking or persistent process injection mechanisms designed to maintain execution and expand malware control across the environment. During execution, the malware established network communication with the external IP address `43[.]128[.]54[.]184` over port 1234, suggesting command-and-control (C2) or remote communication activity.

The malware's original name is "SandboxieCrypto.exe".

Process Name	Process ...	Protocol	State	Local Port	Remote Address	Remote Port	Create Time	Module Name
svchost.exe	2324	TCP	Established	50789	43.128.54.184	1234	5/29/2025 3:14:35 PM	svchost.exe
mitmweb.exe	1120	TCP	Syn Sent	50790	43.128.54.184	1234	5/29/2025 3:14:35 PM	mitmweb.exe

8:18:5...	SandboxieCrypto.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager
8:18:5...	SandboxieCrypto.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager
8:18:5...	SandboxieCrypto.exe	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode
8:18:5...	SandboxieCrypto.exe	CreateFile	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	QueryBasicInfor...	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	CloseFile	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	CreateFile	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	CreateFileMapp...	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	CreateFileMapp...	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	Load Image	C:\Windows\System32\msvcrt.dll
8:18:5...	SandboxieCrypto.exe	Thread Create	
8:18:5...	SandboxieCrypto.exe	Load Image	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll
8:18:5...	SandboxieCrypto.exe	Load Image	C:\Windows\System32\sechost.dll
8:18:5...	SandboxieCrypto.exe	Thread Create	
8:18:5...	SandboxieCrypto.exe	CloseFile	C:\Users\vboxuser\Desktop\View Details\SbieDll.dll

External Threat Landscape Management

This campaign indicates the emergence of a highly adaptive and stealth-focused malware operation leveraging government impersonation, multi-region phishing infrastructure, and advanced in-memory execution techniques to target victims through socially engineered tax-related lures.

The campaign demonstrates a broader operational capability beyond a single geography, as the same infrastructure was observed hosting both Indian and Japanese government-themed phishing templates, suggesting the existence of a centralized phishing framework capable of dynamically localizing content for different target regions.

The use of trusted third-party email delivery services, spoofed government identities, and multilingual phishing portals highlights a deliberate effort to increase delivery success rates and bypass traditional email security controls. From a technical perspective, the malware ecosystem reflects a mature threat model emphasizing defense evasion, runtime obfuscation, and covert persistence through DLL Search Order Hijacking, reflective PE loading, dynamic API hashing, token manipulation, and WebSocket-based encrypted C2 tunnelling.

The absence of hardcoded infrastructure, combined with proxy-aware communication and memory-resident execution, indicates an operational design intended to reduce forensic visibility and complicate attribution efforts. Additionally, the presence of Chinese-language artifacts, reusable phishing infrastructure, and modular payload staging suggests possible access to shared malware development resources or professionally maintained builder frameworks commonly observed within organized cybercriminal ecosystems.

Collectively, these characteristics indicate an evolving external threat landscape in which threat actors increasingly combine localized social engineering, modular malware delivery, and stealth-oriented post-exploitation techniques to maintain resilient and difficult-to-detect intrusion capabilities against both enterprise and individual targets.

Conclusion

This investigation identified a highly sophisticated multi-stage malware campaign delivered through an Indian tax-themed phishing operation leveraging social engineering, government impersonation, and ZIP-based payload delivery mechanisms to compromise victims.

The malware architecture demonstrated advanced defense-evasion and anti-analysis capabilities, including DLL Search Order Hijacking, API Hooking, Token Manipulation, Mersenne Twister-based execution logic, COM callback-based execution, Mutated RC4 encryption, Reflective PE Loading, Control Flow Flattening (CFF), and WebSocket-based Command-and-Control communication. The combination of memory-resident execution, dynamic API resolution, runtime obfuscation, and stealth-oriented communication techniques significantly reduces forensic visibility and complicates detection through conventional security mechanisms.

Additionally, the observed multi-region government impersonation activity, Chinese-language artifacts, and reusable phishing infrastructure suggest a flexible and modular operational framework; however, attribution remains at moderate confidence. Overall, the findings indicate a stealth-focused, highly modular, and technically mature malware ecosystem designed to maintain persistent and covert access while minimizing opportunities for detection and analysis.

RECOMMENDATIONS

Strategic Recommendations

- **Strengthen Security Awareness Programs:** Conduct regular security awareness training focused on social engineering, government impersonation tactics, phishing identification, and malicious attachment handling to improve user resilience against deception-based attacks.
- **Implement Advanced Email and Threat Intelligence Controls:** Deploy email security gateways, threat intelligence feeds, and domain reputation monitoring solutions capable of identifying spoofed domains, phishing infrastructure, and suspicious multi-region impersonation campaigns before they reach end users.

Tactical Recommendations

- **Deploy Enhanced Detection Rules:** Implement YARA, Sigma, and EDR detection rules to monitor indicators associated with DLL Search Order Hijacking, API Hooking, Reflective PE Loading, token manipulation, and WebSocket-based C2 communication.
- **Strengthen Network Monitoring and Filtering:** Configure network intrusion detection systems (IDS) and proxy monitoring solutions to detect suspicious HTTP/1.1 protocol upgrades, WebSocket communication patterns, and unusual outbound traffic associated with dynamic C2 tunnelling behaviour.

Operational Recommendations

- **Enable Continuous Endpoint Monitoring:** Establish continuous EDR monitoring and memory analysis capabilities to identify memory-resident malware behaviour, process injection activities, and anomalous runtime API usage that may bypass traditional antivirus solutions.
- **Establish Incident Response and Threat Hunting Procedures:** Develop dedicated incident response playbooks and perform periodic threat hunting exercises focused on phishing-based intrusions, malicious DLL execution, token abuse, and multi-stage malware execution chains to reduce detection and response time.

APPENDIX 1

MITRE ATT&CK MAPPING

Tactic	ID	Technique Name	Description
Initial Access	T1566.002	Phishing: Spear phishing Link	Threat actors used social engineering emails containing malicious links redirecting victims to phishing infrastructure hosting malware payloads.
Initial Access	T1189	Drive-by Compromise	Victims were redirected to a malicious phishing webpage which delivered a ZIP package containing staged malware payloads.
Execution	T1204.002	User Execution: Malicious File	Execution required user interaction through extraction and execution of कर विवरण.exe.
Execution / Defense Evasion	T1574.001	Hijack Execution Flow: DLL	Malware used DLL Search Order Hijacking to force loading of malicious SbieDll.dll instead of the legitimate library.
Execution	T1620	Reflective Code Loading	The malware manually reconstructed and executed PE components entirely in memory without using LoadLibrary().
Defense Evasion	T1027	Obfuscated/Compressed Files and Information	Malware used Control Flow Flattening, mutated RC4, and dynamic execution logic to conceal behaviour.
Defense Evasion	T1140	Deobfuscate/Decode Files or Information	Encrypted payloads within SbieDll.bin were decrypted during runtime before execution.

Defense Evasion	T1036	Masquerading	The executable कर विवरण.exe used deceptive naming and impersonation techniques to appear legitimate.
Defense Evasion	T1055	Process Injection	Malware used reflective loading and memory-based execution mechanisms to inject and execute payloads without standard process creation behaviour.
Defense Evasion / Privilege Escalation	T1134	Access Token Manipulation	Hooks targeting SetThreadToken() and GetTokenInformation() manipulated security tokens and impersonation contexts.
Credential Access / Defense Evasion	T1553	Subvert Trust Controls	Malware abused trusted Sandboxes DLL mechanisms and legitimate components for execution concealment.
Discovery	T1082	System Information Discovery	Malware performed operating system version checks and environment validation before execution.
Command and control	T1071.001	Application Layer Protocol: Web Protocols	Malware established C2 communication through HTTP/WebSocket protocols.
Command and control	T1573	Encrypted Channel	Communication channels were protected using encrypted WebSocket-based sessions.
Command and control	T1090	Proxy	Presence of HTTP CONNECT functionality indicates support for proxy-aware communication through enterprise environments.
Command and control	T1105	Ingress Tool Transfer	Additional payload components were dynamically loaded and transferred between execution stages.

INDICATORS OF COMPROMISE (IOCs)

S. No	Indicators of Compromise (IOCs)	Type	Remarks
1	guhxmng.com	Domain	Block

2	naiqja.icu	Domain	Block
3	zh-welcome-1xbet.com	Domain	Block
4	d.pc-weide.com	Subdomain	Block
5	taxations.cn-web-okooo.com	Subdomain	Block
6	taxations.indiagov.it.com	Subdomain	Block
7	zhengfu666.com	Domain	Block
8	asdqxcdsa.icu	Domain	Block
9	appradarr.cc	Domain	Block
10	ws4962.com	Domain	Block
11	185b7a487316454da04e9cc0fe6eb370bb2955cf6096fe3e8c02c46f8989ba37	SHA-256	Block
12	4c9061a07d667bf7dd6f597a43a8552af2f4277b7be06d6ea138abdb668d6a49	SHA-256	Block
13	949acbe543fc244ffbc981ea169067da7c5792af3c3d19b2c31b3d7e19106880	SHA-256	Block
14	be31a63cad112723178289968ad6f93a576c5a7984099c42eec3521cdf6e5fc0	SHA-256	Block
15	7d87a86dbd2379ef2516c99258137cd9c25ca19c48aeb096c5332c02fcbf16d0	SHA-256	Block
16	43.128.54.184	IP Address	Block

YARA RULES

import "hash"

rule Operation_Tax_Shadow_Malware_Campaign

{

meta:

author = "Cyfirma Research"

description = "Detection rule for phishing infrastructure and malware samples"

date = "2026-05-29"

strings:

// Domains

\$d1 = "guhxmng.com" nocase

\$d2 = "naiqja.icu" nocase

\$d3 = "zh-welcome-1xbet.com" nocase

\$d4 = "d.pc-weide.com" nocase

\$d5 = "taxations.cn-web-okooo.com" nocase

\$d6 = "taxations.indiagov.it.com" nocase

\$d7 = "zhengfu666.com" nocase

\$d8 = "asdqxcdsa.icu" nocase

\$d9 = "appradarr.cc" nocase

\$d10 = "ws4962.com" nocase

// IP Address

\$ip1 = "43.128.54.184"

condition:

// Match embedded infrastructure indicators

any of (\$d*) or

\$ip1 or

// Match known malware hashes

hash.sha256(0, filesize) ==

"185b7a487316454da04e9cc0fe6eb370bb2955cf6096fe3e8c02c46f8989ba37" or

hash.sha256(0, filesize) ==

"4c9061a07d667bf7dd6f597a43a8552af2f4277b7be06d6ea138abdb668d6a49" or

hash.sha256(0, filesize) ==

"949acbe543fc244ffbc981ea169067da7c5792af3c3d19b2c31b3d7e19106880" or

hash.sha256(0, filesize) ==

"be31a63cad112723178289968ad6f93a576c5a7984099c42eec3521cdf6e5fc0" or

hash.sha256(0, filesize) ==

"7d87a86dbd2379ef2516c99258137cd9c25ca19c48aeb096c5332c02fcbf16d0"

}

Copyright CYFIRMA. All rights reserved.