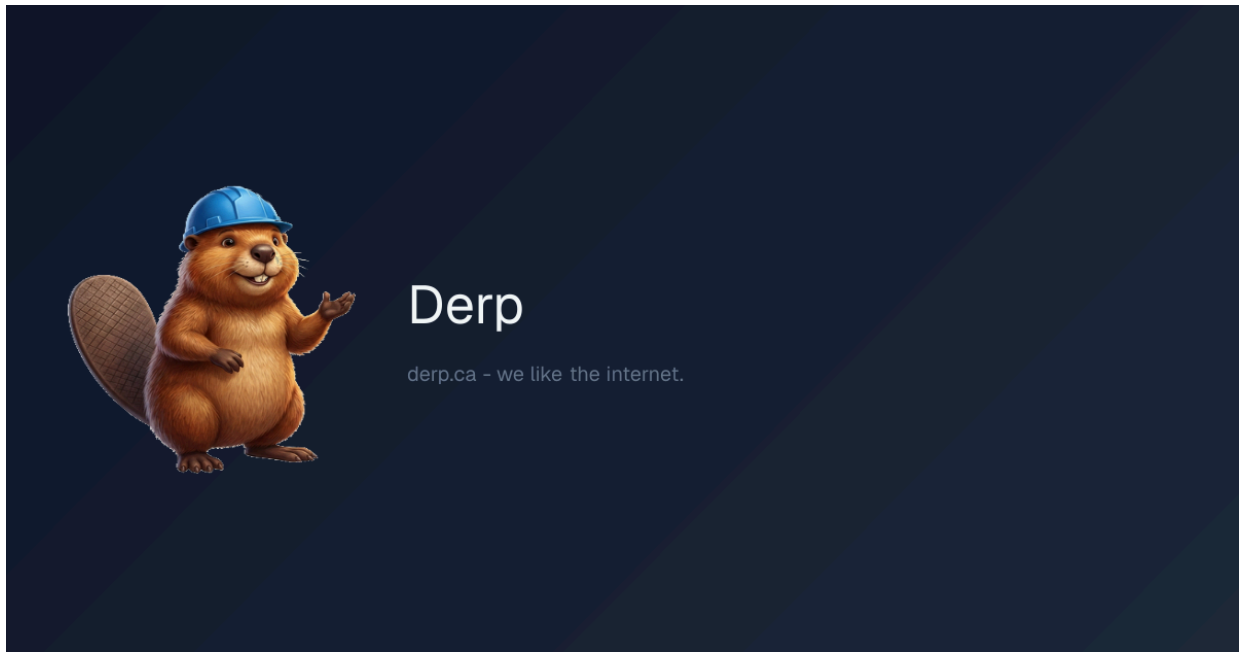


VECT ransomware: small files decrypt, large files lose their nonces

 derp.ca/research/vect-ransomware-nonce-loss

Kirk

May 2, 2026



malware ransomware vect reverse-engineering encryption chacha20

VECT looks like ransomware. This Windows build also looks like a file-format accident.

For this `vect.exe` sample, small files are recoverable. The binary contains a static ChaCha20 file key, and each small encrypted `.vect` file keeps the 12-byte nonce needed to replay the cipher stream. Pair the recovered key with the appended trailer and the encrypted bytes decrypt back to the original content. The validation result is `match=true`.

Large files do not get the same mercy. The routine encrypts four `32 KiB` chunks with four fresh nonces, then appends only the final 12-byte nonce to the file. The first three nonces are overwritten before they ever reach disk.

[Check Point Research published the broader VECT 2.0 analysis](#) on April 28, 2026. Their report says the Windows, Linux, and ESXi lockers use raw ChaCha20-IETF, process files above `131,072` bytes as four chunks, and save only the last 12-byte nonce. This Windows sample reaches the same boundary through observed file writes and a small-file decrypt check.

For ChaCha20, the key is only half the map. The nonce tells the cipher which stream to replay. Lose the nonce and the encrypted bytes are still there, but the route back to the original content is gone.

That is not a negotiation problem. It is a missing-metadata problem.

Public context

[Halcyon described Vect](#) as an emerging ransomware-as-a-service operation that began affiliate recruitment in late December 2025 and showed early public victim claims in January 2026. The actor advertised Windows, Linux, and VMware ESXi support, Tor-based affiliate and victim infrastructure, revenue sharing, Safe Mode execution, network discovery, lateral movement, and GPO propagation.

[ThreatLocker later obtained panel access](#) and documented a working affiliate dashboard with victim profiles, chat, support, payout tracking, and builders for Windows, Linux, and ESXi. The same writeup says the panel listed an exfiltration-only binary as "coming soon" at the time of writing.

[Ransomware.live listed 25 VECT victim claims](#) when checked on May 2, 2026, with the first discovered claim on January 6 and the latest on April 15. Treat those as leak-site and tracker claims. They help with timing and public footprint; intrusion details need separate confirmation.

The TeamPCP context is the access side of the story. [Wiz reported](#) TeamPCP activity after supply-chain compromises involving Trivy, KICS, LiteLLM, and Telnyx, with stolen cloud, SSH, Kubernetes, and CI/CD secrets validated and used in victim environments. [Unit 42 also described](#) the TeamPCP supply-chain campaign against security and developer infrastructure.

That does not make every TeamPCP data claim a VECT locker deployment. It means the access pipeline around VECT is more worrying than the locker code deserves.

[Halcyon's later recovery warning](#) says not to assume payment can restore affected files. The Windows sample below shows why, byte by byte.

Sample overview

Field	Value
Sample	<code>vect.exe</code>
Triage task	260429-1zdvjjas2w
Submitted	2026-04-29T09:57:00Z
SHA256	8ee4ec425bc0d8db050d13bbff98f483fff020050d49f40c5055ca2b9f6b1c4d
SHA1	f4b904fb6ba8474cb87f26302b74c4b82c106003
MD5	207b1a60f803d348c795d382f5aed9c3
Type	PE32+ Windows x64 console executable

Field	Value
Size	1,453,056 bytes
Compile timestamp	2026-02-13T14:45:09Z
Branding	VECT 2.0
Extension	.vect
Ransom note	!!!_READ_ME_!!!.txt

The binary's strings expose an operator interface:

Switch	Meaning
-h, --help	help
-v, --verbose	verbose output
-p, --path <dir>	target a specific path
-c, --creds <b64>	override credentials
--gpo / --no-gpo	toggle GPO spread
--mount / --no-mount	toggle network mount behavior
--stealth / --no-stealth	toggle self-delete
--force-safemode	force safe-mode boot behavior

The --help path also exposed embedded default credential material:

Field	Value
Base64	TEFCXEFkbWluaXN0cmF0b3I6UEBzc3cwcmQxMjMh
Decoded	LAB\Administrator:P@ssw0rd123!

Windows behavior

Two controlled executions completed cleanly: one targeted folder and one default run.

Run	Arguments	Result	Dropped files	Note counter
Targeted folder	--path C:\Users\michael\Documents\backups	ExitProcess(0)	3	Files encrypted: 3
Default	none	ExitProcess(0)	14	Files encrypted: 14

Observed impact behavior includes:

Area	Behavior
Defender	PowerShell <code>Set-MpPreference</code> disables realtime, behavior, IOAV, and script scanning
Shadow copies	<code>vssadmin delete shadows /all /quiet</code>
Services	service-control activity involving <code>vss</code> and <code>MSSQLSERVER</code>
Event logs	clears <code>Application</code> , <code>Security</code> , <code>System</code> , and <code>Windows</code> PowerShell with <code>wevtutil cl</code>
Registry	SafeBoot entries, Run key path, and <code>DisableTaskMgr</code> policy value
Marker	creates <code>C:\ProgramData\.vect</code> ; Halcyon reports the same path as a version-specific pre-launch exit marker
Wallpaper	writes <code>dvm3_wall.bmp</code> , SHA256 <code>f5bfd20eda559537e560cd31409f2345afd8de92a41d40beb1ff3064779615c0</code>
Cleanup	delayed self-delete through <code>cmd /c ping 127.0.0.1 -n 3 >nul & del /f /q ...</code>

The embedded PowerShell is built for lateral movement and remote execution. It references admin-share copy to `\\$pc\C$\ProgramData\$.name`, scheduled tasks, CIM/WMI process creation, PowerShell remoting, and remote service creation through `sc.exe`.

The controlled environment showed the hidden PowerShell GPO/lateral command launch path. Real propagation would need reachable domain peers, shares, or hosts.

Ransom note and contact

The recovered ransom note is 1,656 bytes before impact counters are updated.

Field	Value
Note SHA256	b9f4c4bad3a8262ade99cfb6785df34afe8aff9a2652026d2350238e265ed458
Banner	VECT
Claimed cipher	ChaCha20
Unique ID	5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f
Chat	vectordntlcr1mfkcm4alni734tbcrnd5lk44v6sp4lqal6noqrgnbyd[.]onion/chat/5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f
Qtox	1A51DCBB33FBF603B385D223F599C6D64545E631F7C870FFEA320D84CE5DAF076C1F94100B5B

File impact

The relevant file routine starts at 0x14006a4b0. It renames the target to a .vect path, opens the file, reads chunks into a shared buffer, encrypts those chunks through 0x140068db0, writes the encrypted bytes back, and then appends one 12-byte trailer.

Representative examples from the default execution:

Original	Encrypted path
quarterly-planning.docx	quarterly-planning.docx.vect
large-archive.zip	large-archive.zip.vect
mail-store.pst	mail-store.pst.vect
server-backup.tibx	server-backup.tibx.vect
vm-snapshot.pvhd	vm-snapshot.pvhd.vect

The same execution also produced malformed edge-case names such as `customer-list.csvvect` and `customer-list.csvvvect`. Those are useful as observed file artifacts, but `.vect` is the primary extension behavior.

The byte-level evidence comes from watching what the malware wrote to encrypted files. One example:

Field	Value
Path	c:\users\michael\desktop\quarterly-planning.docx.vect
Write offset	0
Write size	32,768 bytes
Write SHA256	784cc502265b71c77d2e4bdb2408e6b5b05ea508b5640b0154d16151ef631d7b
Encryption path	0x14006a8ac
Trailer appended at end of file	093db13cddec7e65f6de5b05

For `large-archive.zip.vect`, the routine wrote four encrypted 32 KiB chunks at offsets 0, 0x80000, 0x100000, and 0x180000, then wrote one 12-byte trailer at offset 0x200000.

Why small files decrypt

The sample's ransom note says ChaCha20. The code path and file writes agree.

The encrypt/decrypt routine at 0x140068db0 passes the file key into a ChaCha implementation. The ChaCha state uses:

Field	Value
Constant	expand 32-byte k
Counter	0
Nonce	fresh 12 bytes per encrypted chunk
Core rounds	10 double-rounds
Rotate constants	16, 12, 8, 7

The file key is static in the binary. The encrypted-file format is simple: encrypted bytes plus a per-chunk nonce. Recovery depends on the static key and the correct nonce for the chunk being decrypted.

The decrypt check proves the small-file path. Using the captured `quarterly-planning.docx.vect` encrypted chunk and its 12-byte trailer, it decrypts the chunk back to the matching original bytes:

Field	Value
Nonce / trailer	093db13cddec7e65f6de5b05
Ciphertext SHA256	784cc502265b71c77d2e4bdb2408e6b5b05ea508b5640b0154d16151ef631d7b
Plaintext SHA256	5ff074ddad88b7fcb4339cb7a3e68341061792869e43673b2de8525a75476bd8
Decrypted SHA256	5ff074ddad88b7fcb4339cb7a3e68341061792869e43673b2de8525a75476bd8
Match	true

That is the clean half of the story. For a file that fits in one encrypted chunk, the appended nonce is enough.

The large-file break

Large files are different.

For `large-archive.zip.vect`, the final file is exactly original size plus 12 bytes:

Field	Value
Original size	2,097,152 bytes
Final size	2,097,164 bytes
Extra data	one 12-byte trailer
Final trailer	644f9183a0e96634ac85445a

The decrypt check tested that persisted trailer against each encrypted large-file chunk:

Offset	Result with final trailer
0	match=false
0x80000	match=false
0x100000	match=false
0x180000	match=true

The final trailer belongs to the final encrypted chunk only.

The code explains why. In large-file mode, the routine computes a stride from `file_size >> 2`, runs four read/encrypt/write iterations, and calls `0x140068db0` each time. Each call generates a fresh 12-byte nonce into the same trailer buffer. The first three nonces are overwritten before any trailer is written. After the loop, the code seeks to the end of the file and writes exactly `0x0c` bytes from the current trailer buffer.

The observed file format has no place for the missing three nonces.

Where the missing nonces would have to be

A disk-only recovery path for large files would need the first three nonces to be stored somewhere else. I checked the places they would realistically appear:

Channel checked	Result
File tail	exactly one 12-byte trailer
Bytes before trailer	normal file bytes
Sidecar files	only the encrypted targets, notes, marker, and wallpaper
Registry	SafeBoot, Run key, and <code>DisableTaskMgr</code> values
Network surface	share/mount enumeration imports

The disk-recovery boundary is:

File class	Disk-only recovery with this key and trailer format
<code><= 0x20000</code> bytes	assuming this build's static key is available, the final trailer is sufficient to decrypt the encrypted body
<code>> 0x20000</code> bytes	only the final encrypted chunk is decryptable from disk alone

Full large-file restoration would require capturing the transient per-chunk nonces while the malware is running, for example around `0x140068db0`.

About the public possible decryptor

A public post from DarkWebInformer raised the obvious question: is there already a VECT ransomware decryptor?

Source	Link
Tweet	DarkWebInformer status
Repository	DarkWebInformer/vect-ransomware-decryptor

That question is what sent me into the sample. The analysis above was done from the Windows VECT binary and its encrypted-file behavior first. Only after that did I compare the public repository against the recovered file format.

The repository README says the tool was found on Telegram, describes it as possibly a VECT decryptor, says the author did not have an encrypted file to test, and recommends running it only in an isolated environment.

The Go source targets a different file format than the Windows VECT sample analyzed here.

The file-format mismatch is structural:

Area	Public repo	This VECT sample
Extension	<code>.vect1</code>	<code>.vect</code>
Cipher mode	<code>chacha20poly1305.New()</code> / AEAD	raw ChaCha20-compatible stream encryption
Auth tag	expects a 16-byte Poly1305 tag	encrypted bytes plus a 12-byte trailer

Area	Public repo	This VECT sample
Nonce location	expects the nonce at the start of the file	one 12-byte nonce appended at the end of the file
Small-file threshold	0x100000 block logic	0x20000 / 131,072-byte boundary
Large-file behavior	decrypts one leading block, then pads and reassembles	four 32 KiB encrypted chunks at quarter-file offsets
Key recovery	scans for a key using an AEAD auth oracle	static key plus raw ChaCha20-style output

The repo's key-search approach depends on ChaCha20-Poly1305 authentication succeeding for a candidate key. This sample's format is raw encrypted bytes plus a trailing nonce, and the nonce sits at the opposite end of the file from where the tool expects it.

For small files in this sample, the format is internally consistent: encrypted body plus the final 12-byte nonce. For large files, the sample generates four per-chunk nonces and only preserves the final one. A third-party decryptor cannot recover nonce material that was never written to disk.

This GitHub project may relate to a different `.vect1` artifact, an early proof of concept, a confused Telegram tool, or something else entirely. It targets a different file format than the VECT 2.0 Windows sample analyzed here.

Use the file-format details in this post before trusting any public decryptor claim. If you are working a VECT case and need help validating whether your files match this format, reach out to .

Responder takeaway

For this build, small-file recovery is technically plausible when the static key and the appended 12-byte nonce are available. Large-file recovery from disk alone stops at the final encrypted chunk, the only chunk that keeps its nonce.

Preserve affected files, preserve memory if the process is still live, and plan recovery around backups. A paid decryptor cannot reconstruct nonce material that the locker never wrote down.

What this sample shows

Area	Evidence
Ransomware impact	<code>.vect</code> renames, encrypted chunks, updated note counters
Recovery disruption	<code>vssadmin delete shadows /all /quiet</code>

Area	Evidence
Defense evasion	Defender-disable PowerShell and event-log clearing
Persistence / reboot impact	SafeBoot and Run-key registry paths
Lateral material	hidden GPO/lateral PowerShell launch and embedded remote-execution script strings
Cleanup	delayed self-delete
Wallpaper	dvm3_wall.bmp write

This Windows sample is a VECT-branded ransomware locker with local and lateral impact features. Its file encryption is ChaCha20-compatible, and its small-file format is recoverable. The large-file format loses the data needed to finish the job.

Appendix: cryptographic artifacts

The decrypt check uses these little-endian file-key qwords from 0x140069140:

Qword	Value
1	6a510957f1bf8ece
2	d9aae3aa5e861215
3	2d0abbf111fcd2a3
4	9f02ffe6000f6be8

The resulting file-key SHA256 is:

```
dd2105e5d97add32151d3340022e7bcb2ddefb2c635725a5fea2327c38a2d4bd
```

Detection artifacts

Type	Value
Sample SHA256	8ee4ec425bc0d8db050d13bbff98f483fff020050d49f40c5055ca2b9f6b1c4d
Sample MD5	207b1a60f803d348c795d382f5aed9c3
Ransom note	!!!_READ_ME_!!!.txt
Ransom note SHA256	b9f4c4bad3a8262ade99cfb6785df34afe8aff9a2652026d2350238e265ed458
Extension	.vect
Branding	VECT 2.0
Wallpaper	dvm3_wall.bmp
Wallpaper SHA256	f5bfd20eda559537e560cd31409f2345afd8de92a41d40beb1ff3064779615c0
File key SHA256	dd2105e5d97add32151d3340022e7bcb2ddefb2c635725a5fea2327c38a2d4bd
Chat onion	vectordntlcrImfkcm4alni734tbcrnd5lk44v6sp4lqal6noqrgnbyd[.]onion
Unique ID	5cb9f0f9-e171-403f-bed9-a3cd6ce36d1f
Qtox	1A51DCBB33FBF603B385D223F599C6D64545E631F7C870FFEA320D84CE5DAF076C1F94100B5B
Command	powershell -Command "Set-MpPreference -DisableRealtimeMonitoring \$true - DisableBehaviorMonitoring \$true -DisableIOAVProtection \$true - DisableScriptScanning \$true"
Command	vssadmin delete shadows /all /quiet
Command	wevtutil cl "Application"
Command	wevtutil cl "Security"

Type	Value
Command	<code>wevtutil cl "System"</code>
Command	<code>wevtutil cl "Windows PowerShell"</code>
Self-delete	<code>cmd /c ping 127.0.0.1 -n 3 >nul & del /f /q "C:\Users\michael\AppData\Local\Temp\vect.exe"</code>
Embedded credential lead	<code>TEFCXEFkbWluaXN0cmF0b3I6UEBzc3cwcmQxMjMh</code>
Remote copy string	<code>\\\$pc\C\$\ProgramData\ \$name</code>
Remote execution string	<code>Invoke-CimMethod -CimSession \$sess -ClassName Win32_Process -MethodName Create</code>
Remote service string	<code>sc.exe \\\$pc create \$svc binPath= "C:\ProgramData\ \$name"</code>

Research note

This kind of work gets better with more telemetry. If you operate a threat intelligence platform with API access and can provide a researcher account, please reach out to .

VECT does not need a dramatic ending. The operator-facing story says ransomware. The file format says something harsher: for large files, the sample throws away three of the four nonces it needs, then asks the victim to trust a decryptor that cannot have the missing pieces.

Kirk

I like the internet. Want to get in touch?