

SOC INSIGHTS

UNCOVERING THREE NEW
ATTACK VECTORS & THE
UNDERLYING TRUST GAP:
DEF CON 2026
FIELD REPORT



OVERWATCH

EXECUTIVE SUMMARY

Last month, Sentinel Blue's Overwatch team attended DEF CON 2026 to bring back actionable intelligence and early warnings on attack patterns headed toward DIB environments. Three distinct attack vectors surfaced across three very different layers of the modern enterprise.

1. **Prompt Injection is the New SQL Injection:** Attackers are exploiting the implicit trust that AI agents place in retrieved content, allowing them to turn ordinary tickets, emails, or documents into a path to unauthorized tool calls and database access.
2. **Implicit Trust & Logic Bugs:** Individually authorized, low-privilege actions are being chained together to produce unauthorized outcomes like denial of service without any single step tripping a signature-based detection.
3. **Hitting the Devs in the Software Supply Chain:** Attackers are bypassing hardened production environments by instead targeting developers and popular open-source packages, turning a single compromised account into a self-propagating compromise with wide-reaching impacts.

While the mechanics differ, all three vectors point to the same underlying truth: Regardless of whether an organization is running a legacy OS, cloud infrastructure, or the latest AI tooling, the primary opportunity for attacks will always be misconfigured controls that offer too much trust.

This report profiles each attack vector individually and offers recommendations for defending against each.

ATTACK VECTOR 1: PROMPT INJECTION IS THE NEW SQL INJECTION

For years, injection attacks worked by tricking a query parser into treating attacker-supplied data as a command. Today, AI agents sit inside the trust boundary, reading content, taking action, and giving the old attack pattern a new home.



How It Works

Prompt injections happen in a simple chain. Nothing in that chain requires a sophisticated exploit. It only requires that the agent trust its input.

Untrusted text arrives

A ticket body, an email, a web page, a vendor doc



The agent reads it

Instructions and data share one channel



A tool call fires

The agent composes the query and runs it



The database answers

With whatever privileges the agent holds

For the DIB, expect to see more prompt injection attacks in the coming months. AI copilots and agents are increasingly wired into ticketing systems, internal documentation, and databases across defense contractor environments. An agent that can read a support ticket and query a database is an agent that can be manipulated into querying that same database on an attacker's behalf.

Your Three Defensive Priorities

Sanitize the pipeline, not the prompt. Treat every retrieved document and tool response as hostile input. Remember that a system prompt is guidance, not a command from on high.

Limit access. An agent reaches only as far as you allow it, so use read-only roles, purpose-built service principals, and row-level limits. Assume the query will eventually be attacker-chosen.

Log the action, not the chat. Detection lives at the tool layer: the query issued, the rows returned, the endpoint touched. Prompt transcripts on their own will not rebuild an incident.



ATTACK VECTOR 2: IMPLICIT TRUST & LOGIC BUGS

For years, the dominant Windows vulnerability class was the memory bug. Now, attackers are increasingly turning to logic bugs: flaws that come from individually authorized, low-privilege actions that are being chained together to produce unauthorized outcomes. Because every individual step is authorized, no single link in the chain reads as a bug or trips a signature. The attack happens entirely through behavior that's strung together in a way no single control is designed to catch, leading to consequences like denial of service, privilege escalation, or persistence.

How It Works

Like prompt injections with AI agents inside your trust boundary, this vector isn't a failure of any single control. In this case, the risk comes from authorized actions being evaluated only on an individual basis and not as a sequence. Here's the breakdown.

Find a primitive

This is a simple action a SYSTEM service will perform for a standard user. On its own it looks harmless, which is exactly why it survives review.



Chain them together

Every step is authorized, so no single step reads as a bug. Each link avoids tripping a signature; the chain itself is the exploit.



Land the impact

Denial of service, privilege escalation, or persistence, reached entirely through documented behavior.

For the DIB, this will defeat the tooling that many contractors are using. There's nothing for signature-based detection to flag, since every individual step is legitimate, documented Windows behavior. An environment can be fully patched against known vulnerabilities and still be exposed to this pattern, because there's no CVE for "using a feature exactly as designed, in the wrong order."



Example: WerSVC, the Windows Error Reporting Service

- The service accepts a client request without checking whether the caller is authorized to make it.
- A standard user can aim the error reporting feature at a process ID of their choosing.
- The targeted service stops. That is a denial of service from an unprivileged account.
- Same class of flaw, publicly documented as CVE-2026-20817: an authorization gap in the same service, reached over ALPC.

Your Three Defensive Priorities

Hunt behavior instead of signatures. Be on the lookout for things like odd parent and child process pairs and service stops that nobody scheduled.

Keep up your patch cadence. Microsoft closed the gap by removing the feature outright, for example.

Turn off what you don't need. CIS baselines already call for disabling this service.

ATTACK VECTOR 3: HITTING THE DEVS IN THE SOFTWARE SUPPLY CHAIN

Because production security has gotten harder to breach directly, attackers are going around it and aiming instead for the people and pipelines that build the software running in production in the first place. Threat actors are now targeting developers and popular open-source packages, turning a single compromised account into a self-propagating compromise with wide-reaching impacts.

How It Works

Monitoring usually stops after the first tier in the supply chain, but a worm that republishes itself doesn't. Here's what that chain can look like.

A maintainer's account is compromised

Phishing, credential theft, or a leaked token targets the person who can publish, not the code itself.



Malicious code ships under a trusted identity

A trojanized version is published from the real account, often carrying valid signatures or provenance since the build pipeline itself was never touched.



Downstream projects pull it in automatically

Every project depending on that package (even several layers removed) installs the malicious version the next time it builds.



The payload harvests credentials on the host

Cloud keys, CI secrets, and additional publishing tokens are exfiltrated from any environment that installed it.



Stolen tokens republish the payload into other packages

Finally, the compromise spreads on its own, reaching packages and organizations with no direct relationship to the original account.

For the DIB, DFARS 252.204-7012 requires flow-down of security obligations to subcontractors handling covered defense information. But it's easy to overlook the open-source dependency tree sitting underneath your build pipeline. Additionally, a package can carry a valid, signed provenance attestation and still be malicious, because the attestation only proves that the build process and not the underlying source was legitimate.

Your Three Defensive Priorities

Know what you actually run. An SBOM is your best friend here to know what you pull in, what that pulls in behind it, and who holds publish rights on each package.

Harden the developers. Phishing-resistant MFA on every registry and repository identity. Reviewed, source-controlled publish paths. Short-lived tokens instead of standing ones.

Treat it as an ongoing effort. Supply chain risk assessment should be an ongoing program rather than a one-time review.



RECOMMENDATIONS & CONCLUSION

None of these three attack vectors represent a new category of vulnerability so much as a new set of places for an old problem to hide. Misconfigured and misplaced trust is the constant here, whether it's in an AI agent's input, in a Windows service, or in a maintainer's account.

For the DIB, the solution doesn't lie in a particular tool or a single patch. Instead, it's ongoing discipline that requires you to ask the same questions regardless of your environment.

- **Know what you're running.** An SBOM for your dependencies, an inventory of what your AI agents can touch, and a clear picture of which services and features are actually needed will all help you understand exactly what's in your environment and where the risks lie.
- **Least-privilege access, always.** Whether it's an AI agent's credentials, a standard user's access to a Windows service, or a maintainer's publish rights on a package, assume that access will eventually be used by an attacker and scope it accordingly.
- **Look at behavior holistically.** None of these three attack vectors will trip a traditional signature. Detection processes need to examine everything holistically: the tool call, the process chain, the lifecycle script, and beyond.
- **Treat everything as ongoing, not one-time.** Supply chain risk, AI governance, and system hardening all merit a standing program, not an annual checkbox.

DEF CON 2026 was an excellent reminder that attack vectors will keep multiplying across every layer of the stack, from AI agents to operating systems to the open-source code sitting underneath them. Luckily, so will the defenders watching for them. DIB companies that adopt the ongoing discipline of asking what's running, who has access, and what looks different will be far better positioned to catch the next vector before it lands.

For more information, [explore our other SOC insights and threat intelligence reports.](#)



QUESTIONS?

Contact Elizabeth Cory at elizabeth.cory@sentinelblue.com



SENTINELBLUE