

Eight bytes to SYSTEM in EA Javelin

2026-10-16 · Miles N.

A signed installer handed Everyone the right to choose what a LocalSystem service runs.

Affected	1.0.14789537.0
Fixed	1.0.15166423.0
CWE	CWE-732
CVE	CVE-2026-96542

Windows services carry their own permissions. Setting them is the least interesting thing an installer does.

EA Javelin’s signed installer set one that let Everyone change the configuration of a service running as LocalSystem. Any local user could pick the program, and Windows would then run it as `NT AUTHORITY\SYSTEM`. A later build added that one right to four of the service's five access-control entries.

I started with the goal of creating a reproducible Ludus range for human-driven, LLM-assisted vulnerability research. EA Javelin Anticheat was not the obvious thing to throw the kitchen sink at, but I was curious. I expected plenty of the unpacking work to fail. That was part of the point. Every failed approach should leave behind something useful for the next packed sample. That can be a future post.

The range

Two Windows VMs, both defined in one Ludus config file. The split follows the work. Javelin installs a Windows service, so one VM ran Javelin and stayed as the installer left it. I read the live state there with `sc.exe` and `reg.exe`. The other VM was the workbench, and it's Windows on purpose. Decoding a service descriptor and walking a packed PE both want the real Windows API, not a reimplementation I could get wrong. It held the analysis databases, one record per build, which is what let me line the builds up later. Codex ran in its own container and reached both over SSH. I used RDP when a step needed a real logged-in session. I kept a ledger of what I was asking next.

A VM rebuild was one command, so whenever I doubted the target's state, I rebuilt it instead of reasoning about it. The evidence lived on the workbench, so I could wipe the target for free. I picked the wrong binary first. GameService is a 60,014,840-byte signed DLL: seven sections whose names all begin with packer, one of them about 44.5 MB of code, an entry point in one packer section that jumps straight into another, a 4 KB read-write-execute page with nothing behind it on disk. An automated unpacking service called it "Likely Not Packed" and returned nothing.

Most of what I tried on it went nowhere.

Two evenings in, I knew a lot more about the protection and still had no clean image.

After the second evening, I ran out of ideas and asked an open-ended question:

```
next work?
```

That is quoted as typed, and so is every other prompt of mine here. Cleaning it up would flatter the record.

GPT-5.6 Sol recommended I stop banging my head against the wall and instead inventory the installed Windows state. It ran the equivalent of:

```
sc.exe qc EAAntiCheatService
sc.exe sdshow EAAntiCheatService
reg.exe query HKLM\SYSTEM\CurrentControlSet\Services\EAAntiCheatService /s
```

The Javelin files under Program Files had ordinary protected ACLs. The service object did not.

The service object

`EAAntiCheatService` is created stopped, demand-start, and running as LocalSystem. `sc.exe sdshow EAAntiCheatService` returns its security descriptor as one line:

```
D:(A;;CCDCLCRPWPCRSDRC;;;WD)(A;;CCDCLCSWRPWPDTLOCRRC;;;SY)(A;;CCDCLCSWRPWPDTLOC
```

`D:` opens the DACL, which is who may do what. `S:` opens the SACL, which is what gets audited and grants nobody anything. Broken up, the DACL is five entries, one per principal:

Entry	Applies to
`(A;;CCDCLCRPWPCRSDRC;;;WD)`	Everyone
`(A;;CCDCLCSWRPWPDTLOCRRC;;;SY)`	LocalSystem
`(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)`	Administrators
`(A;;CCDCLCSWLOCRRC;;;IU)`	Interactive users
`(A;;CCDCLCSWLOCRRC;;;SU)`	Service logon users

The first one is the problem. `WD` is Everyone, and `DC` is `SERVICE_CHANGE_CONFIG`. The Interactive users and Service logon users entries carry `DC` too.

Windows keeps a service’s executable path in the service configuration, which is what that right governs. So a correctly ACLed binary under Program Files doesn't help. Nothing requires an attacker to touch it.

A candidate is not SYSTEM

A descriptor only tells you what the SCM advertises, nothing about what actually runs. The first probe carried no real payload: a genuine standard-user token, open the service for `SERVICE_CHANGE_CONFIG`, write the existing demand-start value back unchanged. It worked. That established the primitive and stopped there.

I pulled the model back before it got ahead of the evidence:

> we also haven't test it out, let's not jump the cart

It had already assigned a CVSS score before anything ran. I struck that from the ledger and tested impact for real: same token, same config-write path, but now the executable pointed at a fixed marker program, and I requested a start.

```
identity_name=NT AUTHORITY\SYSTEM
identity_sid=S-1-5-18
is_system=True
```

`StartServiceW` returned 1053. The model logged that as a clean success. It wasn't. The marker was never a real service and had no control dispatcher registered, so SCM had nothing to wait on. It launched the binary anyway. The marker wrote its SYSTEM identity to disk. SCM reported the failure only after its start timed out. The 1053 is just SCM noticing, late, that nobody answered its phone. By then the code had already run.

A standard user picked the code, and Windows ran it as SYSTEM.

On a machine left in that state, `SERVICE_CHANGE_CONFIG` doesn't care which standard user holds the token. Any interactive logon qualifies. It matters most where one machine runs a Javelin-protected game under many non-admin users: LAN centers and internet cafes, school and library labs, shared family PCs. This path never touches an EA file, never writes to Program Files, never bypasses game integrity, and never goes near Javelin's memory.

Where the descriptor comes from

Finding the permission took one check. Explaining it took three days.

The service does not exist before installation. The installer that creates it is validly signed by Electronic Arts, reports version 1.0.14789537.0, and its service-creation path reads a stored SDDL string and applies it with `SetServiceObjectSecurity`. The stored string resolves to the live DACL, with the same entries in the same order.

EA also ships a game-local installer, 1.0.12317156.0. Its Everyone ACE differs in one place:

```
- older  (A;;CC  LCRPWPCRSDRC;;;WD)
+ newer  (A;;CCDCLCRPWPCRSDRC;;;WD)
```

`DC` goes into four of the five DACL entries: Everyone, LocalSystem, Interactive users, and Service logon users. Administrators already carried it, so four entries change, not five. Each insertion is two ASCII bytes. The installer stores the DACL as a null-terminated SDDL string, and that string grows from `0x87` to `0x8F`. Count the DACL printed above and you get 142 characters, 143 with the terminator, eight bytes over the older build's 135.

The older Everyone ACE still decodes to delete, start, and stop rights on a service running as LocalSystem.

Reading one of these yourself

The string looks worse than it is. Every entry is six fields separated by semicolons, and three of them are usually empty:

```
(A;;CCDCLCRPWPCRSDRC;;;WD)
 |  |                 |
 |  |                 who it applies to
 |  what they are allowed to do
 A for allow, D for deny
```

The middle field is a run of two-character tokens, read left to right. On a service object, these are the ones worth knowing:

```
CC  query the configuration
DC  change the configuration   <- picks the executable
LC  query status
SW  enumerate dependent services
RP  start
WP  stop
DT  pause and continue
LO  interrogate
CR  send a user-defined control
SD  delete
RC  read the permissions
WD  change the permissions
WO  take ownership
```

And the last field is who:

```
WD  Everyone
IU  anyone logged on interactively
AU  Authenticated Users
BU  Users
BA  Administrators
SY  LocalSystem
```

The tokens are Directory Service names reused by numeric value, so `DC` means delete-child on a directory object and change-config on a service. Same two letters, different meaning depending on what you are looking at, and nothing in the string tells you which. And `WD` is Everyone in that last field, but `WRITE_DAC` in the rights field.

To read your own, on any Windows machine:

```
sc.exe sdshow <servicename>
```

Anything granting `DC`, `SD`, `WD`, or `WO` to `WD`, `IU`, `AU`, or `BU` is worth a second look.

That is the quick pass, and it has four blind spots. A principal with no alias prints as a raw SID, so a domain group like `S-1-5-21-…-1105` slides straight past a check that only reads two-letter names. The rights field can be a hex mask instead of tokens, and `0x7800003F` matches none of the letters above. `GW` grants change-config and `GA` grants all of them, and neither spells out a letter. And a deny entry earlier in the list can cancel an allow that appears later, so a string can look bad and not be. None of the four can be read off the letters.

I checked every access-right claim above against Microsoft's documentation before I submitted it and again before publication. Service Security and Access Rights is the page with the full table plus the default descriptor Windows gives a new service. It states this post's whole point outright: granting access rights to untrusted users "such as SERVICE_CHANGE_CONFIG or SERVICE_STOP" can "allow them to interfere with the execution of your service, and possibly allow them to run applications under the LocalSystem account." ACE Strings has every token if you hit one I have not listed.

Asking the models to attack the proof

GPT-5.6 Sol at medium reasoning handled the range work and the first impact proof. I moved to high reasoning for the deeper installer custody work. Before submission, I opened a fresh session, handed Daybreak Blue the accumulated evidence, and gave it one instruction:

```
now verify you or the previous models haven't hallucinated or cheated
```

It found two things, and I checked both myself before changing anything. One was the `StartServiceW` wording above. The other mattered more: an offline string decoder had used a constant from the wrong installer family. I withdrew the static regression claim, reopened both analysis databases, recovered the per-build constants again, and restored it only after two independent decoders reproduced both descriptors and the same eight-byte difference.

Neither finding touched the privilege chain.

The models earned two credits: the pivot to the installed state, and the bad decoder constant. They also wrote conclusions before evidence.

Fix and disclosure

No non-administrative principal should hold `SERVICE_CHANGE_CONFIG` on a LocalSystem service. The update path has to repair existing installations, because hardening only fresh service objects leaves already affected endpoints where they were.

The regression test is one line: from a genuine standard-user token, `OpenServiceW` for `SERVICE_CHANGE_CONFIG` must fail with `ERROR_ACCESS_DENIED`, and keep failing after repair, reinstall, and authenticated launch.

EA shipped it. They replaced the CDN object with build 1.0.15166423.0. On 1 September, I brought up a fresh Ludus VM and ran two paths from one snapshot. First, the patched build on a clean machine. Then I rolled back to the affected build to reproduce the vulnerable descriptor, and applied the patched installer on top of it. Both produce:

```
(A;;CCLCRPWPCRSDRC;;;WD)
```

`DC` is gone, a standard-user token is denied `SERVICE_CHANGE_CONFIG`, and applying the patched installer over an affected descriptor rewrites it. That token was interactively logged on, so the Interactive-users entry no longer grants change-config either.

That ACE is the older build, so the same token still opens the service for start, stop, and delete, all with error 0. None of those restore the chain, because choosing the executable needs change-config.

Liu and the folks from EA’s Security team were a pleasure to work with. They triaged it quickly and kept me updated throughout. The replacement object I tested reported an August 24 Last-Modified date, thirteen days after I reported it. In my upgrade test, applying the patched installer over the affected build rewrote the bad descriptor. It repairs

endpoints that are already affected, which is the case that matters for remediation.

TLDR: I spent days building workflows to unpack the hardest binary in the system. Windows answered the first security question with an access check.

Big shout out to Erik Hunstad, who built Ludus at Bad Sector Labs. I would not be doing all the cool things I am doing nearly as quickly without it.

AI DISCLOSURE

This research was done with AI assistance, as described above. I ran and verified every test myself.

/ᐠ · ⌣ · ᐟ\ · quality control: the cat