Una vulnerabilidad de alta gravedad en Google Dialogflow CX habría permitido que un atacante con permisos de edición sobre un agente habilitado con Code Blocks comprometiera otros agentes que utilizaran la misma funcionalidad dentro de un mismo proyecto de Google Cloud.

A partir de ese acceso inicial, el atacante habría podido visualizar conversaciones activas, obtener la información compartida por los usuarios e incluso hacer que los chatbots enviaran mensajes maliciosos, como solicitudes para volver a introducir contraseñas.

La empresa de ciberseguridad Varonis descubrió esta vulnerabilidad, denominada Rogue Agent. El problema afectaba únicamente a las organizaciones que desarrollaban agentes mediante los Playbooks de Dialogflow junto con Code Blocks personalizados, una característica que permite ejecutar código Python propio. No se trataba de una vulnerabilidad explotable de forma remota ni por usuarios no autenticados.

Para aprovecharla era necesario contar con el permiso *dialogflow.playbooks.update* sobre al menos uno de estos agentes, por lo que el escenario de ataque más probable involucraba a un empleado malintencionado o a una cuenta de desarrollador comprometida, más que a un atacante externo. Sin embargo, una vez obtenido ese acceso, el alcance podía extenderse a todos los agentes del mismo proyecto.

Google corrigió la vulnerabilidad y tanto Varonis como la propia compañía afirmaron que no existen evidencias de que haya sido explotada en ataques reales.

Un único archivo modificable controlaba los Code Blocks de todos los agentes

Los Code Blocks de Dialogflow permiten incorporar código Python personalizado dentro del flujo conversacional de un chatbot para validar entradas, controlar su comportamiento o ejecutar herramientas definidas por el desarrollador. Dicho código se ejecuta en un entorno de Cloud Run administrado por Google y compartido entre todos los agentes que utilizan Code Blocks dentro del mismo proyecto de Google Cloud.

Aunque Google administra completamente ese entorno y los clientes no tienen acceso ni capacidad de control sobre él, Varonis descubrió que no existía un aislamiento efectivo entre los distintos agentes.

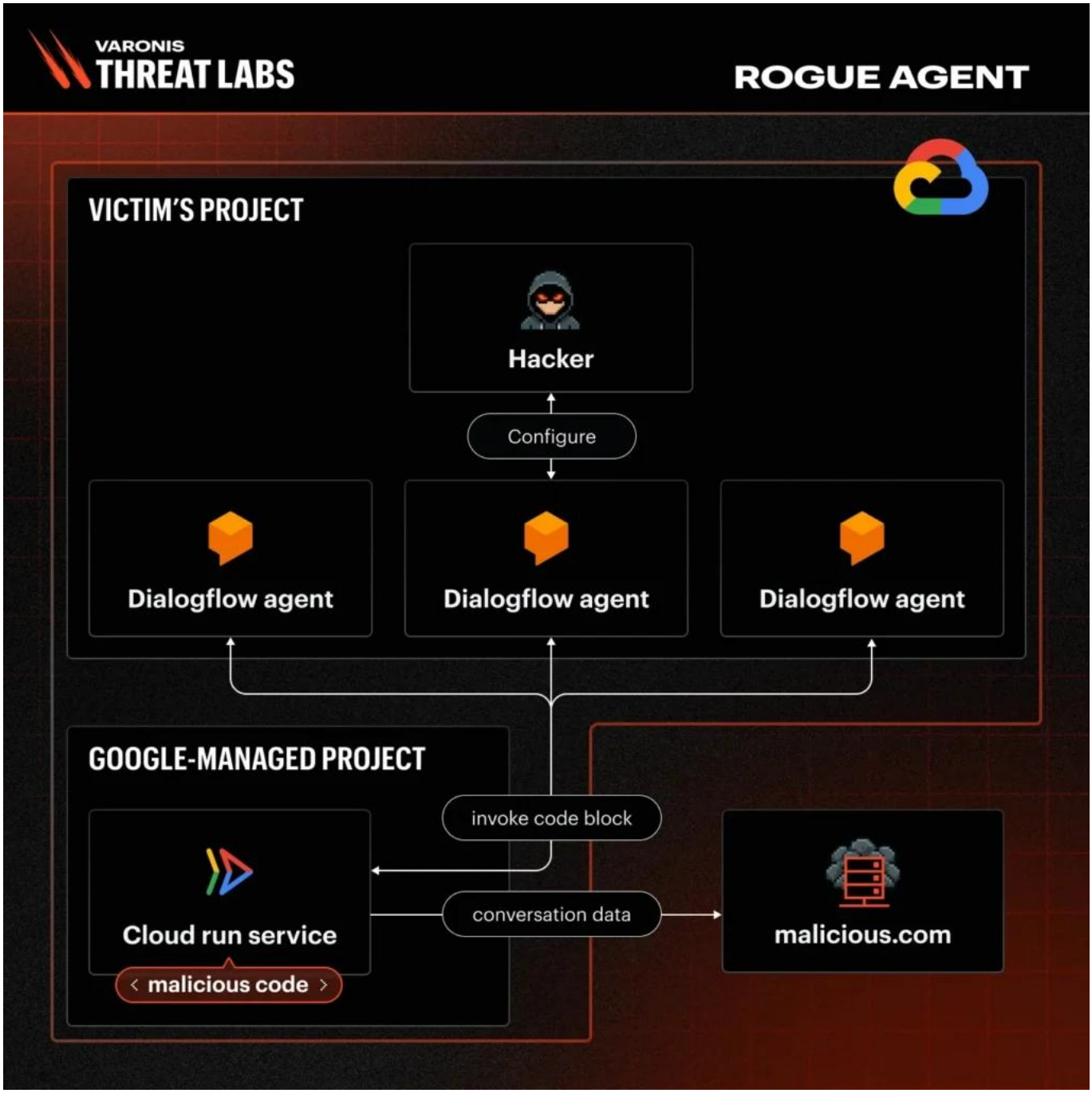
Cada vez que un agente ejecutaba un Code Block, el código del desarrollador se añadía a un código interno de inicialización y posteriormente se ejecutaba mediante la función de Python *exec()*. Ese código preparaba las variables y funciones disponibles para el bloque.

Entre las variables se encontraban el historial completo de la conversación y el estado de la sesión, incluyendo el identificador de sesión. También estaban disponibles funciones como *respond()*, encargada de generar las respuestas enviadas por el chatbot.

Durante el análisis, Varonis identificó que el archivo *code_execution_env.py*, responsable de preparar este entorno de ejecución, se encontraba dentro del entorno compartido con permisos de escritura.

Debido a ello, un único Code Block podía reemplazar dicho archivo. El procedimiento consistía en descargar una versión modificada de *code_execution_env.py* desde un servidor controlado por el atacante y sobrescribir el archivo original dentro del contenedor en ejecución.

A partir de ese momento, la versión maliciosa pasaba a ejecutarse durante todas las ejecuciones posteriores de Code Blocks en cualquier agente que compartiera ese entorno. El código obtenía exactamente los mismos privilegios que el código legítimo, pudiendo acceder al historial de conversaciones, al estado de las sesiones y a la función *respond()*.



Esto permitía capturar todas las conversaciones, enviarlas silenciosamente a un servidor externo y hacer que el chatbot publicara mensajes controlados por el atacante. Uno de los

escenarios descritos consistía en una campaña de phishing donde el bot solicitaba al usuario volver a verificar sus credenciales, permitiendo al atacante capturar cualquier dato introducido.

Para dificultar la detección, el atacante restauraba posteriormente el Code Block original desde la consola de Dialogflow. Aunque la interfaz mostraba nuevamente el código legítimo, el archivo sobrescrito seguía ejecutándose dentro del contenedor sin que el cliente pudiera detectarlo.

El entorno aislado presentaba otras dos debilidades

Varonis también notificó dos problemas adicionales que no requerían sobrescribir archivos.

En primer lugar, el entorno de ejecución de Code Blocks disponía de acceso irrestricto a Internet. Mediante la biblioteca estándar *urllib*, los investigadores consiguieron enviar información directamente a servidores externos y recibir instrucciones desde ellos.

Según Varonis, este comportamiento permitía eludir los mecanismos de *VPC Service Controls*, diseñados por Google Cloud para impedir la salida de información desde servicios protegidos. Como el entorno de ejecución se encontraba fuera de dicho perímetro, podía comunicarse libremente con Internet, convirtiéndose en un canal tanto para la exfiltración de datos como para el control remoto.

La segunda vulnerabilidad, considerada de menor impacto, consistía en que el entorno tenía acceso al *Instance Metadata Service (IMDS)*, un servicio interno encargado de proporcionar credenciales de la infraestructura en la nube.

Al consultar dicho servicio se obtenía un token perteneciente a una cuenta de servicio administrada por Google.

Aunque esa cuenta disponía de privilegios limitados, los investigadores señalaron que un entorno de ejecución destinado a código aislado no debería poder acceder al IMDS bajo

ninguna circunstancia.

La actividad apenas dejaba rastros

La modificación del archivo ocurría dentro del entorno administrado por Google, al que los clientes no tienen visibilidad. Además, Cloud Logging no registraba ni el cambio realizado sobre el archivo ni el código inyectado.

Como consecuencia, detectar este comportamiento desde la perspectiva del cliente resultaba complejo, aunque no imposible. No obstante, algunas de las acciones necesarias para preparar el ataque sí dejaban ciertos indicios aprovechables durante una investigación.

Varonis comunicó la vulnerabilidad a través del Programa de Recompensas por Vulnerabilidades de Google en noviembre de 2025. Google publicó una primera mitigación en abril de 2026 y completó la corrección definitiva en junio de 2026, aproximadamente siete meses después de recibir el informe. No se asignó ningún identificador CVE.

Qué revisar si utilizabas Code Blocks

Si antes de la corrección utilizabas agentes de Dialogflow CX con Playbooks basados en Code Blocks y deseas comprobar si pudiste haber sido objetivo de este ataque, el primer paso consiste en revisar los permisos de acceso.

El permiso *dialogflow.playbooks.update* constituye el punto de entrada principal de la vulnerabilidad, por lo que conviene identificar cuidadosamente qué usuarios y roles disponen de él.

Asimismo, se recomienda:

- Revisar los registros de auditoría *DATA_WRITE* de la API de Dialogflow para identificar modificaciones inesperadas en los Playbooks y correlacionarlas con usuarios, direcciones IP u horarios de acceso inusuales.

- Ejecutar consultas en Cloud Logging sobre solicitudes fallidas de usuarios, ya que los mensajes de error podrían revelar excepciones provocadas por Code Blocks maliciosos.
- Acceder a la consola de Dialogflow, revisar los Playbooks de cada agente y verificar que todos los Code Blocks presentes correspondan únicamente a implementaciones previamente autorizadas.

Un tipo diferente de vulnerabilidad en sistemas de IA

Gran parte de las vulnerabilidades recientes relacionadas con inteligencia artificial se han basado en manipular directamente el comportamiento del modelo mediante técnicas como la inyección de instrucciones.

Investigaciones anteriores de Varonis, como Reprompt y SearchLeak, demostraron cómo un solo clic podía facilitar el robo de información en Microsoft Copilot. Del mismo modo, Noma Security presentó ForcedLeak, una técnica que ocultaba instrucciones dentro de formularios web de Salesforce para extraer datos de sistemas CRM.

Por otra parte, investigadores de Microsoft demostraron que la inyección de *prompts* podía evolucionar hasta lograr ejecución de código dentro del framework Semantic Kernel.

Sin embargo, Rogue Agent no atacaba al modelo de IA directamente. En su lugar, explotaba una funcionalidad legítima para desarrolladores junto con un entorno de ejecución compartido e invisible, accesible únicamente mediante un permiso de edición aparentemente inofensivo.

En un escenario de estas características, un permiso que aparenta permitir únicamente modificar contenido equivale realmente a un permiso de ejecución de código. Cualquier usuario autorizado para añadir un Code Block puede ejecutar código Python arbitrario dentro de un entorno compartido que el cliente no tiene posibilidad de inspeccionar.

Por ello, los permisos de edición de agentes deberían gestionarse con el mismo nivel de

protección que los controles sobre el entorno de ejecución. Incluso cuando el proveedor confirme que el problema ha sido corregido, los clientes continúan sin disponer de mecanismos para inspeccionar internamente dicho entorno.