HijackLoader Expands Techniques to Improve Defense Evasion

By Donato Onofri - Emanuele Calvelli

Archived: 2026-04-05 18:00:35 UTC

- HijackLoader continues to become increasingly popular among adversaries for deploying additional payloads and tooling
- A recent HijackLoader variant employs sophisticated techniques to enhance its complexity and defense evasion
- CrowdStrike detects this new HijackLoader variant using machine learning and behavior-based detection capabilities

CrowdStrike researchers have identified a HijackLoader (aka IDAT Loader) sample that employs sophisticated evasion techniques to enhance the complexity of the threat. HijackLoader, an increasingly popular tool among adversaries for deploying additional payloads and tooling, continues to evolve as its developers experiment and enhance its capabilities.

In their analysis of a recent HijackLoader sample, CrowdStrike researchers discovered new techniques designed to increase the defense evasion capabilities of the loader. The malware developer used a standard process hollowing technique coupled with an additional trigger that was activated by the parent process writing to a pipe. This new approach has the potential to make defense evasion stealthier.

The second technique variation involved an uncommon combination of process doppelgänging and process hollowing techniques. This variation increases the complexity of analysis and the defense evasion capabilities of HijackLoader. Researchers also observed additional unhooking techniques used to hide malicious activity. This blog focuses on the various evasion techniques employed by HijackLoader at multiple stages of the malware.

HijackLoader Analysis

Infection Chain Overview

The HijackLoader sample CrowdStrike analyzed implements complex multi-stage behavior in which the first-stage executable (`streaming_client.exe`) deobfuscates an embedded configuration partially used for dynamic API resolution (using `PEB_LDR_DATA` structure without other API usage) to harden against static analysis.

Afterward, the malware uses WinHTTP APIs to check if the system has an active internet connection by connecting to `https<:>//nginx<.>org` . If the initial connectivity check succeeds, then execution continues, and it connects to a remote address to download the second-stage configuration blob. If the first URL indicated below fails, the malware iterates through the following list:

- `https[:]//gcdnb[.]pbrd[.]co/images/62DGoPumeB5P.png?o=1`
- `https[:]//i[.]imgur[.]com/gyMFSuy.png;`
- `https[:]//bitbucket[.]org/bugga-oma1/sispa/downloads/574327927.png`

Upon successfully retrieving the second-stage configuration, the malware iterates over the downloaded buffer, checking for the initial bytes of a PNG header. It then proceeds to search for the magic value `C6 A5 79 EA`, which precedes the `XOR` key (`32 B3 21 A5` in this sample) used to decrypt the rest of the configuration blob.

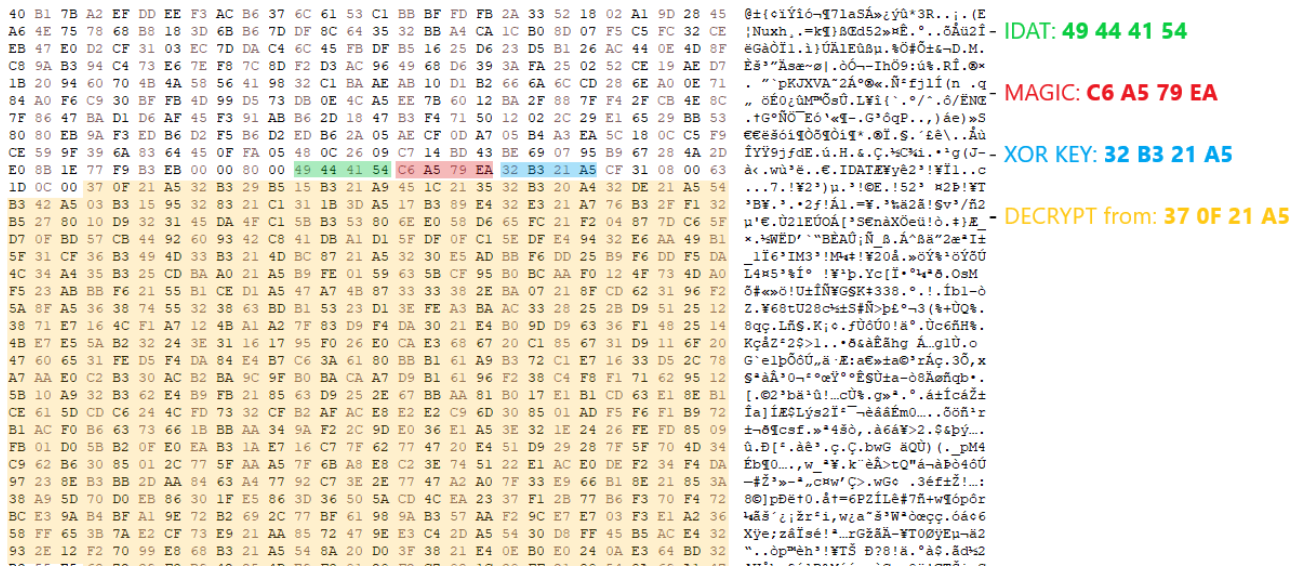

Figure 1. HijackLoader key retrieving and decrypting (click to enlarge)

Following XOR decryption, the configuration undergoes decompression using the `RtlDecompressBuffer` API with `COMPRESSION_FORMAT_LZNT1`. After decompressing the configuration, the malware loads a legitimate Windows DLL specified in the configuration blob (in this sample, `C:\Windows\SysWOW64\mshtml.dll`).

The second-stage, position-independent shellcode retrieved from the configuration blob is written to the `.text` section of the newly loaded DLL before being executed. The HijackLoader second-stage, position-independent shellcode then performs some evasion activities (further detailed below) to bypass user mode hooks using Heaven's Gate and injects subsequent shellcode into `cmd.exe`.The injection of the third-stage shellcode is accomplished via a variation of process hollowing that results in an injected hollowed `mshtml.dll` into the newly spawned `cmd.exe` child process.

The third-stage shellcode implements a user mode hook bypass before injecting the final payload (a Cobalt Strike beacon for this sample) into the child process `logagent.exe`. The injection mechanism used by the third-stage shellcode leverages the following techniques:

- `Process Doppelgänging` Primitives: This technique is used to hollow a `Transacted Section` (`mshtml.dll`) in the remote process to contain the final payload.
- `Process/DLL Hollowing`: This technique is used to inject the fourth-stage shellcode that is responsible for performing evasion prior to passing execution to the final payload within the transacted section from the previous step.

Figure 2 details the attack path exhibited by this HijackLoader variant.

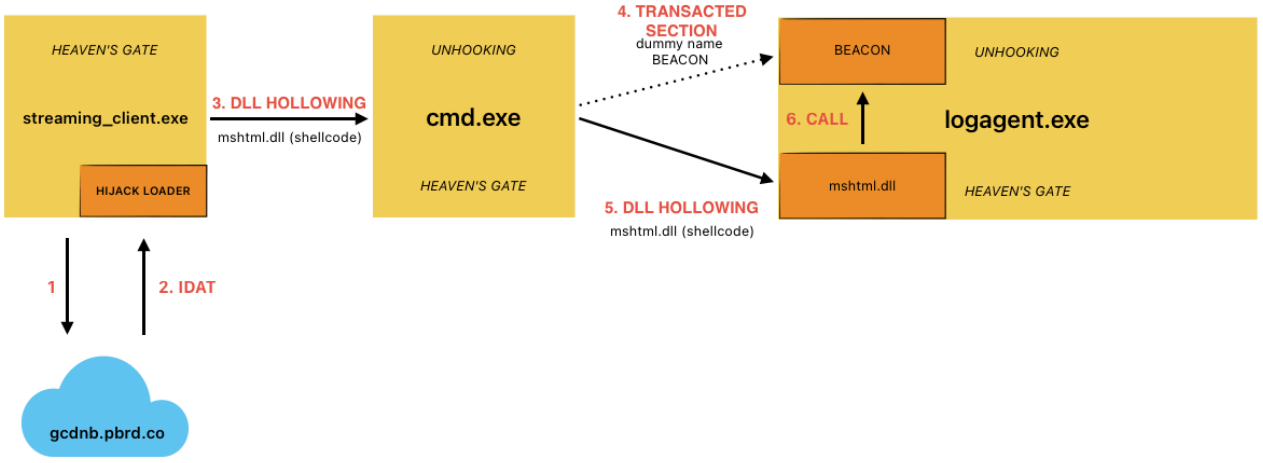

Figure 2. HijackLoader — infection chain (click to enlarge)

Main Evasion Techniques Used by HijackLoader and Shellcode

The primary evasion techniques employed by HijackLoader include hook bypass methods such as Heaven's Gate and unhooking by remapping system DLLs monitored by security products. Additionally, the malware implements variations of process hollowing and an injection technique that leverages transacted hollowing, which combines the transacted section and process doppelgänging techniques with DLL hollowing.

Hook Bypass: Heaven's Gate and Unhooking

Like other variants of HijackLoader, this sample implements a user mode hook bypass using Heaven's Gate (when run in `SysWOW64`) — this is similar to existing (`x64_Syscall` function) implementations.

This implementation of Heaven's Gate is a powerful technique that leads to evading user mode hooks placed in `SysWOW64 ntdll.dll` by directly calling the syscall instruction in the x64 version of `ntdll`.

Each call to Heaven's Gate uses the following as arguments:

- The syscall number
- The number of parameters of the syscall
- The parameters (according to the syscall)

This variation of the shellcode incorporates an additional hook bypass mechanism to elude any user mode hooks that security products may have placed in the x64 `ntdll`. These hooks are typically used for monitoring both the x32 and x64 `nt dll`.

During this stage, the malware remaps the `.text` section of x64 `ntdll` by using Heaven's Gate to call `NtWriteVirtualMemory` and `NtProtectVirtualMemory` to replace the in-memory mapped `ntdll` with the `.text` from a fresh `ntdll` read from the file `C:\windows\system32\ntdll.dll`. This unhooking technique is also used on the process hosting the final Cobalt Strike payload (`logagent.exe`) in a final attempt to evade detection.

Process Hollowing Variation

To inject the subsequent shellcode into the child process `cmd.exe`, the malware utilizes common process hollowing techniques. This involves mapping the legitimate Windows DLL `mshtml.dll` into the target process and then replacing its `.text` section with shellcode. An additional step necessary to trigger the execution of the remote shellcode is detailed in a later section.

To set up the hollowing, the sample creates two pipes that are used to redirect the `Standard Input` and the `Standard Output` of the child process (specified in the aforementioned configuration blob, `C:\windows\syswow64\cmd.exe`) by placing the pipes' handles in a `STARTUPINFOW` structure spawned with `CreateProcessW` API.

One key distinction between this implementation and the typical "standard" process hollowing can be observed here: In standard process hollowing, the child process is usually created in a suspended state. In this case, the child is ***not*** *explicitly created in a suspended state, making it appear less suspicious. Since the child process is waiting for an input from the pipe created previously, its execution is hanging on receiving data from it. Essentially, we can call this an interactive process hollowing variation.*

As a result, the newly spawned `cmd.exe` will read input from the `STDIN` pipe, effectively waiting for new commands. At this point, its `EIP` (`Extended Instruction Pointer`) is directed toward the return from the `NtReadFile` syscall.

The following section details the steps taken by the second-stage shellcode to set up the child process `cmd.exe` ultimately used to perform the subsequent injections used to execute the final payload.

The parent process `streaming_client.exe` initiates an `NtDelayExecution` to sleep, waiting for `cmd.exe` to finish loading. Afterward, it reads the legitimate Windows DLL `mshtml.dll` from the file system and proceeds to load this library into cmd.exe as a shared section. This is accomplished using the Heaven's Gate technique for:

- Creating a shared section object using `NtCreateSection`
- Mapping that section in the remote `cmd.exe` using `NtMapViewOfSection`

It then replaces the `.text` section of the `mshtml` DLL with malicious shellcode by using:

- Heaven's Gate to call `NtProtectVirtualMemory` on `cmd.exe` to set `RWX` permissions on the .text section of the previously mapped section `mshtml.dll`
- Heaven's Gate to call `NtWriteVirtualMemory` on the DLL's `.text` section to stomp the module and write the third-stage shellcode

Finally, to trigger the execution of the remote injected shellcode, the malware uses:

- Heaven's Gate to suspend (`NtSuspendThread`) the remote main thread
- A new `CONTEXT` (by using `NtGetContextThread` and `NtSetContextThread`) to modify the `EIP` to point to the previously written shellcode
- Heaven's Gate to resume (`NtResumeThread`) the remote main thread of `cmd.exe`

However, because `cmd.exe` is waiting for user input from the `STDINPUT` pipe, the injected shellcode in the new process isn't actually executed upon the resumption of the thread. The loader must take an additional step:

- The parent process `streaming_client.exe` needs to write (`WriteFile`) `\r\n` string to the `STDINPUT` pipe created previously to send an input to `cmd.exe` after calling `NtResumeThread` . This effectively resumes execution of the primary thread at the shellcode's entry point in the child process `cmd.exe` .

Interactive Process Hollowing Variation: Tradecraft Analysis

We have successfully replicated the threadless process hollowing technique to understand how the pipes trigger it. Once the shellcode has been written as described, it needs to be activated. This activation is based on the concept that when a program makes a syscall, the thread waits for the kernel to return a value.

In essence, the interactive process hollowing technique involves the following steps:

- **CreateProcess:** This step involves spawning the `cmd.exe` process to inject the malicious code by redirecting `STDIN` and `STDOUT` to pipes. Notably, this process isn't suspended, making it appear less suspicious. Waiting to read input from the pipe, the `NtReadFile` syscall sets its main thread's state to `Waiting` and `_KWAIT_REASON` to `Executive` , signifying that it's awaiting the execution of kernel code operations and their return.
- **WriteProcessMemory:** This is where the shellcode is written into the `cmd.exe` child process.
- **SetThreadContext:** In this phase, the parent sets the conditions to redirect the execution flow of the `cmd.exe` child process to the previously written shellcode's address by modifying the `EIP/RIP` in the remote thread `CONTEXT` .
- **WriteFile:** Here, data is written to the `STDIN` pipe, sending an input to the `cmd.exe` process. This action resumes the execution of the child process from the `NtReadFile` operation, thus triggering the execution of the shellcode. Before returning to user space, the kernel is reading and restoring the values saved in the `_KTRAP_FRAME` structure (containing the `EIP/RIP` register value) to resume from where the syscall was called. By modifying the `CONTEXT` in the previous step, the loader hijacks the resuming of the execution toward the shellcode address without the need to suspend and resume the thread, which this technique usually requires.

Transacted Hollowing² (Transacted Section/Doppelgänger + Hollowing)

The malware writes the final payload in the child process `logagent.exe` spawned by the third-stage shellcode in `cmd.exe` by creating a transacted section to be mapped in the remote process. Subsequently, the malware injects fourth-stage shellcode into `logagent.exe` by loading and hollowing another instance of `mshtml.dll` into the target process. The injected fourth-stage shellcode performs the aforementioned hook bypass technique before executing the final payload previously allocated by the transacted section.

Transacted Section Hollowing

Similarly to process doppelgänging, the goal of a transacted section is to create a stealthy malicious section inside a remote process by overwriting the memory of the legitimate process with a transaction. In this sample, the third-

stage shellcode executed inside `cmd.exe` places a malicious transacted section used to host the final payload in the target child process `logagent.exe`. The shellcode uses the following:

- `NtCreateTransaction` to create a transaction
- `RtlSetCurrentTransaction` and `CreateFileW` with a dummy file name to replace the documented `CreateFileTransactedW`
- Heaven's Gate to call `NtWriteFile` in a loop, writing the final shellcode to the file in 1,024-byte chunks
- Creation of a section backed by that file (Heaven's Gate call `NtCreateSection`)
- A rollback of the previously created section by using Heaven's Gate to call `NtRollbackTransaction`

Existing similar implementations have publicly been observed in this project that implements transaction hollowing. Once the transacted section has been created, the shellcode generates a function stub at runtime to hide from static analysis. This stub contains a call to the `CreateProcessW` API to spawn a suspended child process `logagent.exe` (`c50bffbef786eb689358c63fc0585792d174c5e281499f12035afa1ce2ce19c8`) that was previously dropped by `cmd.exe`

under the `%TEMP%` folder. After the target process has been created, the sample uses Heaven's Gate to:

- Read its `PEB` by calling `NtReadVirtualMemory` to retrieve its base address (`0x400000`)
- Unmap the `logagent.exe` image in the `logagent.exe` process by using `NtUnMapViewofSection`
- Hollow the previously created transacted section inside the remote process by remapping the section at the same base address (`0x400000`) with `NtMapViewofSection`

Process Hollowing

After the third-stage shellcode within `cmd.exe` injects the final Cobalt Strike payload inside the transacted section of the `logagent.exe` process, it continues by process hollowing the target process to write the fourth shellcode stage ultimately used to execute the final payload (loaded in the transacted section) in the remote process. The third-stage shellcode maps the legitimate Windows DLL `C:\Windows\SysWOW64\mshtml.dll` in the target process prior to replacing its `.text` with the fourth-stage shellcode and executing it via `NtResumeThread`.

This additional fourth-stage shellcode written to `logagent.exe` performs similar evasion activities to the third-stage shellcode executed in `cmd.exe` (as indicated in the hook bypass section) before passing execution to the final payload.

CrowdStrike Falcon Coverage

CrowdStrike employs a layered approach for malware detection using machine learning and indicators of attack (IOAs). As shown in Figure 3, the CrowdStrike Falcon® sensor's machine learning capabilities can automatically detect and prevent HijackLoader in the initial stages of the attack chain; i.e., as soon as the malware is downloaded onto the victim's machine. Behavior-based detection capabilities (IOAs) can recognize malicious behavior at various stages of the attack chain, including when employing tactics like process injection attempts.

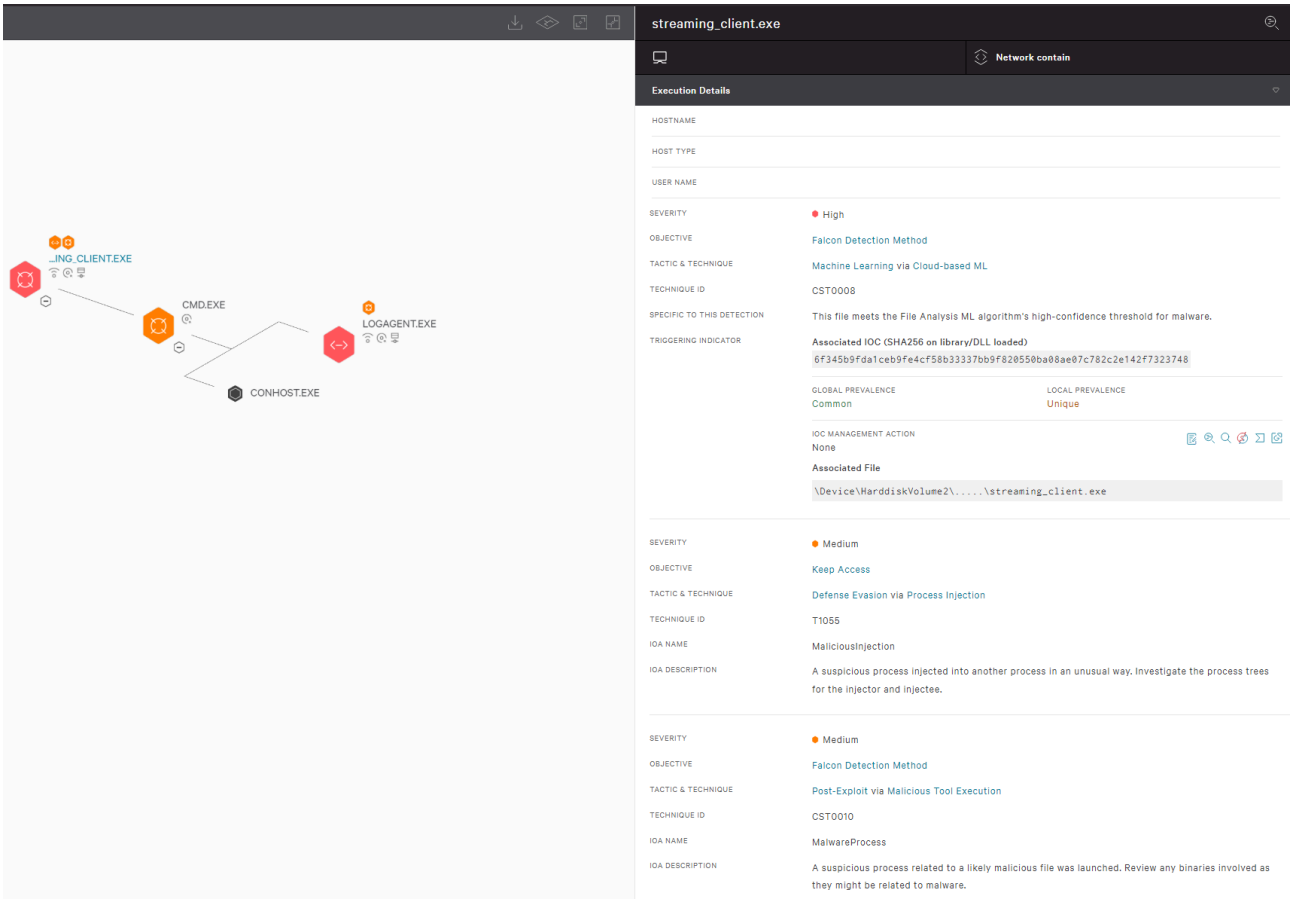

Figure 3. CrowdStrike Falcon platform machine learning and IOA coverage for the HijackLoader sample (click to enlarge)

Indicators of Compromise (IOCs)

MITRE ATT&CK Framework

The following table maps reported HijackLoader tactics, techniques and procedures (TTPs) to the MITRE ATT&CK® framework.

ID	Technique	Description
T1204.002	User Execution: Malicious File	The sample is a backdoored version of `streaming_client.exe`, with the `Entry Point` redirected to a malicious stub.
T1027.007	Obfuscated Files or Information: Dynamic API Resolution	HijackLoader and its stages hide some of the important imports from the IAT by dynamically retrieving `kernel32` and `ntdll` API addresses. It does this by parsing `PEB->PEB_LDR_DATA` and retrieving the function addresses.

T1016.001	System Network Configuration Discovery: Internet Connection Discovery	This variant of HijackLoader connects to a remote server to check if the machine is connected to the internet by using the `WinHttp` API (`WinHttpOpenRequest` and `WinHttpSendRequest`).
T1140	Deobfuscate/Decode Files or Information	HijackLoader utilizes XOR mechanisms to decrypt the downloaded stage.
T1140	Deobfuscate/Decode Files or Information	HijackLoader utilizes `RtlDecompressBuffer` to LZ decompress the downloaded stage.
T1027	Obfuscated Files or Information	HijackLoader drops XOR encrypted files to the `%APPDATA%` subfolders to store the downloaded stages.
T1620	Reflective Code Loading	HijackLoader reflectively loads the downloaded shellcode in the running process by loading and stomping the `mshtml.dll` module using the `LoadLibraryW` and `VirtualProtect` APIs.
T1106	Native API	HijackLoader uses direct syscalls and the following APIs to perform bypasses and injections: `WriteFileW` , `ReadFile` , `CreateFileW` , `LoadLibraryW` , `GetProcAddress` , `NtDelayExecution` , `RtlDecompressBuffer` , `CreateProcessW` , `GetModuleHandleW` , `CopyFileW` , `VirtualProtect` , `NtProtectVirtualMemory` , `NtWriteVirtualMemory` , `NtResumeThread` , `NtSuspendThread` , `NtGetContextThread` , `NtSetContextThread` , `NtCreateTransaction` , `RtlSetCurrentTransaction` , `NtRollbackTransaction` , `NtCreateSection` , `NtMapViewOfSection` , `NtUnMapViewOfSection` , `NtWriteFile` , `NtReadFile` , `NtCreateFile` and `CreatePipe` .
T1562.001	Impair Defenses: Disable or Modify Tools	HijackLoader and its stages use Heaven’s Gate and remap x64 ntdll to bypass user space hooks.
T1055.012	Process Injection: Process Hollowing	HijackLoader and its stages implement a process hollowing technique variation to inject in `cmd.exe` and `logagent.exe` .
T1055.013	Process Injection: Process Doppelgänging	The HijackLoader shellcode implements a process doppelgänging technique variation (transacted section hollowing) to load the final stage in `logagent.exe` .

Additional Resources

- *The CrowdStrike Falcon® platform achieved 100% protection, 100% visibility and 100% analytic detection across all steps in the MITRE Engenuity ATT&CK® Evaluations: Enterprise, Round 5. Learn more in this blog post.*
- *CrowdStrike was named a Leader in the 2023 Gartner® Magic Quadrant™ for Endpoint Protection Platforms — furthest right in Vision and highest in Ability to Execute. Read about it here.*
- *Find out how the Falcon platform stops breaches, saves time and saves money in this IDC analysis: The Business Value of the CrowdStrike Falcon XDR Platform.*
- *See the Falcon platform in action — sign up for a free demo today.*
