1:1 Meetings

Q4 2025

Adarsh Rampup

Setup/lab related POC: Rahul Tandon

Task	Expected Completion date	Status/date
ePO/VM /DLP setup	31/07/2025	✅ Setup ePO/VM /DLP related things done
Jira/Git/Artifactory access	04/08/2025	✅ Got all the access
Dev/ Test/ ONIGMA machine access	04/08/2025	✅ Got access to the systems
Visual studio installation and building the code	04/08/2025	✅ Installed Visual studio on dev machine and building AGENT and OPG repos
Visual Studio remote debugger and windbg debugger	05/08/2025	✅Installing DLP dependencies repos, ✅Syscore and building the code ✅Clipboard remote debugging setup Done
Videos-Sameer's	07/08/2025	✅Done ✅HDLP KT videos video 1 - ePO basics
Product guide-DLP windows	08/08/2025	✅ Sync with siva on flow of keyword blocking in ePO
Confluence pages	08/08/2025	✅ Confluence
Trying out various protection vectors like web, printer, Removable storage, PNP rules	11/08/2025	✅ Done the protection rules
Architecture, Classifications, Registered Document, Email Protection	13/08/2025	✅Product architecture reading ✅HDLP KT videos video 2 - Architecture, Classifications ✅HDLP KT videos video 3 - Registered Document, Email Protection, Network Communication Protection
Query String (URL), Content Classification, Discovery Rules	14/08/2025	✅HDLP KT videos video 4 - Query String (URL), Content Classification, Discovery Rules ⏳HDLP KT videos video 5 - 3rd Party Integrations
Statically linked libs and dynamically linked libs.	15/08/2025	✅Done

<table>
<tr><td>In Linux , .a, for static.
.so for dynamic.
In windows , .dll is dynamic and .lib is static.</td><td></td><td></td></tr>
<tr><td>Dll injection
Hooking into functions
Hook or probe to Agent communication
Attach the debugger
Get the entry point functions for at least two protection vectors like web protection, outlook. Get the call stack and then understand the code flow.
Going through the dll injections, drivers, AFAPR, file filter and other
fcnm
Confluence pages
https://confluence.trellix.com/display/DE/DLPe+Engineering+Home</td><td>18/08/2025 for 4-5 days</td><td>✅Reading the codebase
by Debugged the clipboard workflow</td></tr>
<tr><td>Syscore - hooking implementation for ARM 64</td><td></td><td></td></tr>
<tr><td>Understand GSDK</td><td>25/08/2025</td><td>✅Finished Reading about GSDK from confluence pages
https://confluence.trellix.com/spaces/DE/pages/25557399/Documents+from+google

https://confluence.trellix.com/spaces/DE/pages/280599729/Google+Content+Analysis+SDK+integration+with+Edge+Firefox+and+Island+browsers

https://confluence.trellix.com/spaces/HDLPQA/pages/151519937/GSDK-DLP+Integration

https://confluence.trellix.com/spaces/DE/pages/25557398/Google+Chromium+SDK</td></tr>
</table>

✅Ramp up plan complete

https://jira.trellix.com/browse/DLPW-18791

📄 GSDK keys

GSDK related Tasks

The meeting primarily focused on web protection mechanisms, specifically contrasting the existing web extension approach with the newer Google SDK.

1. Web Protection Methods:

- Ganesh explained that there are two main ways for web protection:
  - **Web Extension:** This is the older method.
  - **Google SDK:** This is a newly developed method by Google.
- The Google SDK was developed last year, and its full features are expected to be released in December with version 114.
- Adarsh noted that Shiva had previously provided a basic overview, mentioning that Google Chrome utilizes the Google SDK for uploads.

2. Google SDK vs. Web Extension (Performance and Conflicts):

- **Web Extension Issues:**
  - The web extension method involves DLL injection and hooking, which often leads to significant performance issues.
  - This method can also cause conflicts and deadlocks with other security software, such as CrowdStrike and SentinelOne.
  - The process is a "long pole" as it involves injecting, inspecting data for confidential content, and then providing a verdict, which is inherently slower.
- **Google SDK Advantages:**
  - The Google SDK directly receives the payload (e.g., uploaded file content).
  - It then directly checks for sensitive content without the need for injection or hooking.
  - This direct approach results in much better performance and avoids conflicts with other security vendors.
  - The goal is to increase adoption of the Google SDK among customers to resolve custom issues related to web protection and performance.

3. Google SDK Configuration:

- **Previous Configuration:**
  - Initially, the Google SDK could only be configured via a CBC (Chrome Browser Cloud Management) and a Chrome management console (a cloud-based management tool).
- **New Configuration Alternatives:**
  - To provide an easier alternative, registry entries can now be set. If these registry entries are configured, it bypasses the need for the Chrome management console.
  - Ganesh also mentioned Group Policy Objects (GPO) as a Windows functionality that allows setting registry entries and performing other operations across an entire enterprise system. Adarsh expressed interest in exploring GPO.

4. Adarsh's Assigned Work:

- **Task:** Adarsh is responsible for setting the registry values for **Google Chrome** browsers.
- **Trigger:** These registry values need to be set when the "Google SDK" option is selected for web protection within the ePO GUI extension.

- **Collaboration:** Shivshankari is currently working on the same task for **Firefox**. Adarsh is advised to connect with her to understand her implementation and then replicate the logic for Edge and Chrome.
- **Registry Value Specificity:** It was clarified that the registry values will be different for Edge and Chrome compared to Firefox. Each browser has specific registries, especially since they support features like Semantic DLP. Ganesh will share these specific registry values with Adarsh.
- **XML Changes:** When an option is selected in the GUI extension, corresponding OPG (Open Policy Group) changes (XML data) are generated and shared with the extension team. Adarsh's code needs to check for this specific XML value. If it's present, the corresponding registry values should be set.
- **Policy Pull and Persistence:**
    - When policies are pulled (e.g., via "Enforce policies" in the Agent Status Monitor or automatically), the system needs to re-check if the Google SDK option is still selected.
    - If the option is selected but the corresponding registry values are not set (e.g., if they were mistakenly removed), the system must re-set them to ensure persistence.

5. Google SDK Installation and Versions:

- The Google SDK is installed automatically as part of the browser installation (Chrome, Edge). It is not part of the agent report or a separate source code.
- Customers on older versions of Chrome or Edge will receive the Google SDK when they update to the latest browser versions.

6. Learning and Development:

- Ganesh emphasized that even though the task might involve some "copy-paste" of code, it will be a valuable exercise for Adarsh to understand the entire development process, including:
    - Coding
    - Building
    - Merging
    - Raising review requests (PRs)
    - Overall code check-in flow.
- Adarsh agreed that this experience would be important for future code changes.

Policy Management

- **Policy Pull and Persistence:** When policies are pulled (e.g., through "Enforce policies" in the Agent Status Monitor or automatically), the system verifies if the Google SDK option is still selected. If selected but registry values are missing, the system re-sets them to ensure persistence.

Google SDK Integration

- **Installation and Versions:** The Google SDK is automatically installed as part of the browser (Chrome, Edge) and is not a separate component in agent reports or source code. Users with older browser versions will receive the SDK upon updating.

Learning and Development

- Ganesh highlighted that this task, despite potential code reuse, offers Adarsh a valuable opportunity to understand the full development lifecycle: coding, building, merging, raising pull requests (PRs), and the overall code check-in process. Adarsh acknowledged the importance of this experience for future code modifications.

From <https://meet.google.com/sez-aopa-eke>

• Technical Foundations for Enterprise Web Protection: A Comparative Analysis and Implementation Guide

Executive Summary

This report provides a comprehensive technical briefing on the fundamental concepts required for the enterprise web protection project. The project's core objective is a strategic migration from a legacy architecture, which relies on browser extensions coupled with Dynamic-Link Library (DLL) injection, to a modern, officially supported framework. This new framework leverages the Google Chrome Content Analysis Connector Software Development Kit (SDK).

The analysis reveals that the new SDK-based method offers superior security, stability, and enterprise manageability. It achieves this by replacing the high-risk, malware-like technique of DLL injection with a sanctioned, API-driven communication protocol. This shift mitigates the performance degradation and software conflicts that characterized the previous solution.

The primary goal of this document is to provide the in-depth knowledge necessary to understand both architectures, master the required enterprise management tools—specifically Chrome Browser Cloud Management (CBCM) and Windows Group Policy Objects (GPO)—and successfully execute the assigned tasks of configuring browser policies and registry values. This report will serve as a foundational learning document, deconstructing each technological component and outlining a clear path for implementation and mastery.

Section 1: Analysis of Web Protection Architectures

The project's context is defined by a critical architectural decision: moving from an intrusive, unstable method of web content inspection to a cooperative, API-driven model. This transition represents a fundamental shift in security philosophy, prioritizing stability and vendor support over the deep but fragile visibility afforded by the legacy approach.

1.1 The Legacy Model: Extension-Based DLL Injection and Hooking

The older method for web protection is a two-part system that operates by forcibly inserting monitoring code into the browser's runtime environment.

Architectural Breakdown

This model consists of a lightweight browser extension and a locally installed native application. The extension acts as an initial sensor, detecting specific web activities such as file uploads or form submissions. Upon detecting a relevant event, the extension signals the native application. This native application then performs the core security function by using DLL injection to load its own code directly into the browser's process memory space (e.g., chrome.exe or msedge.exe).

Mechanism of Action

Once the DLL is injected into the browser's process, it employs a technique known as API hooking. This allows the injected code to intercept system and browser function calls. By hooking functions related to network traffic or file I/O, the DLL can inspect data in transit—for example, the content of a file being uploaded or text being pasted into a form. A key advantage of this method is that it can inspect this data before it is encrypted by Transport Layer Security (TLS), providing deep visibility into user actions. While powerful, this technique is highly intrusive and operates without the browser's explicit consent or knowledge.[1] The meeting notes accurately describe this as the "old web extension method" that "utilizes the older DLL injection and hooking method," which is the root cause of the noted "significant performance and conflicts."

1.2 The Modern Model: The Google Content Analysis Framework

The modern approach replaces the intrusive nature of the legacy model with a formal, cooperative framework designed by browser vendors for the express purpose of content inspection.

De-mystifying the "Google SDK"

The "Google SDK" referenced in the meeting notes is specifically the **Google Chrome Content Analysis Connector SDK**.[3] This is not a generic web protection toolkit but a purpose-built C++ library designed to facilitate a secure and stable integration between the Chrome browser and third-party security agents. The project's primary goal, to check "payloads for sensitive content," aligns directly with the documented purpose of this SDK. The official repository and associated documentation frame its use case entirely around enabling Data Loss Prevention (DLP) agents to scan browser-handled content like file uploads, clipboard pastes, and print jobs for sensitive information.[4] Therefore, the project is fundamentally a DLP initiative, aimed at preventing the unauthorized exfiltration of sensitive corporate data through the browser.

Architectural Breakdown

This model also utilizes a locally installed native agent, but the communication mechanism is fundamentally different and officially sanctioned. The browser, configured via enterprise policy, identifies content that requires analysis. It then sends this content and associated metadata to the local DLP agent using a secure, standardized protocol. The agent performs its analysis and returns a verdict (e.g., allow, block, or warn) to the browser, which then enforces the decision.

Mechanism of Action

The communication between the browser and the native DLP agent is facilitated by **Chrome's Native Messaging API**.[6] This is a formal communication channel that uses the standard input (

stdin) and standard output (stdout) streams of the processes to exchange structured JSON messages. This architecture completely avoids direct memory manipulation, code injection, or API hooking. The browser and the agent operate as two separate, cooperating processes, communicating through a well-defined and secure interface.[7] This directly corresponds to the meeting's description of the "newer Google SDK" which "directly checks payloads for sensitive content without the need for injection or hooking," thereby "avoiding these issues."

Section 2: Deconstructing the Legacy Method: DLL Injection and API Hooking

A thorough understanding of the legacy method's mechanics is essential to appreciate the stability and security benefits of the new architecture. The older approach is being replaced precisely because its foundational techniques are inherently risky and unstable in a modern enterprise environment.

2.1 The Mechanics of DLL Injection

DLL injection is a technique used to force a running process to load and execute code from a Dynamic-Link Library that it was not originally designed to load.[8] This method is frequently used by malicious actors to evade defenses and gain control of legitimate processes.[1]

The Win32 API Sequence

The injection process is typically orchestrated by an external "injector" application that makes a sequence of calls to the Windows API. This sequence manipulates the target process (the browser) from the outside:

OpenProcess(): The injector first obtains a handle to the target process (e.g., chrome.exe), which grants it the necessary permissions to manipulate that process's memory.
VirtualAllocEx(): Using the process handle, the injector allocates a new region of

memory within the target process's virtual address space. This memory region is used to store the path of the DLL to be loaded.
WriteProcessMemory(): The injector writes the full file path of the monitoring DLL (e.g., "C:\Program Files\Vendor\agent.dll") into the memory region allocated in the previous step.
CreateRemoteThread(): This is the critical step where the code execution is triggered. The injector creates a new thread of execution within the target browser process. The starting address for this new thread is cleverly set to the memory address of the LoadLibrary() function, which is a standard Windows function responsible for loading DLLs. The argument passed to this new thread is the memory address where the DLL's path was written. The result is that the browser process is tricked into calling LoadLibrary() on itself, loading the monitoring DLL into its own address space.[1]

Variations and Evasion

More sophisticated versions of this technique exist, such as Reflective DLL Injection and DLL Hollowing (or Module Stomping). These methods write the DLL's raw code directly into memory and manually perform the loading process, bypassing LoadLibrary() entirely to make detection by security software more difficult.[1]

2.2 API Hooking for Content Inspection

Once the monitoring DLL is successfully loaded into the browser's process space, it must intercept relevant operations to inspect data. This is achieved through API hooking.

Purpose and Mechanism

The goal of hooking is to intercept function calls. For instance, to inspect a file upload, the DLL might need to intercept the network send() function. A common method to achieve this is **Import Address Table (IAT) Hooking**. The IAT is a structure within an executable file that lists all the external functions it needs to call from other DLLs. The injected monitoring DLL can modify the browser's IAT in memory, overwriting the address of a legitimate function like send() with the address of its own custom inspection function. When the browser attempts to call send(), it is transparently redirected to the inspection function. This function can then examine the data, and to ensure the browser continues to operate correctly, it will subsequently call the original send() function to complete the operation.[9]

2.3 Inherent Security Risks and Stability Concerns

The fundamental problem with the legacy architecture is that it is built on techniques that are indistinguishable from those used by advanced malware. This creates a model that is inherently in conflict with both the browser it is monitoring and the other security software on the endpoint.

Mimicking Malware

The entire process of DLL injection—using CreateRemoteThread to force code execution in another process—and API hooking are classic techniques used by malware for defense evasion, credential theft, and establishing persistence.[1] Consequently, modern Endpoint Detection and Response (EDR) and antivirus solutions are specifically engineered to detect and block these behaviors. This means the web protection product is in a constant battle with other security tools on the same machine, leading to false positives, blocked functionality, and instability.

The "Long Pole" Problem

The meeting notes refer to this method as a "long pole," an apt analogy for a fragile, single point of failure. The injection and hooking process relies on the internal, undocumented memory structures and function addresses of the browser. When the browser is updated, these internal details can change without notice, causing the hooks to fail and potentially crashing the entire browser. Similarly, when a security vendor like Crowdstrike or SentinelOne updates its agent, its detection heuristics may change, causing it to suddenly identify the web protection agent as malicious and terminate it. This creates a brittle system that requires constant maintenance and is prone to breaking.

Process Conflicts and System Deadlocks

Many security products use their own low-level hooking or injection techniques to monitor system activity. When multiple products attempt to hook the same functions in the same process, they can interfere with each other, leading to race conditions, memory corruption, and in the worst cases, system-wide deadlocks that require a hard reboot.[11] This explains the "conflicts with other security software" mentioned in the meeting. To avoid this, many security vendors explicitly require that their processes be added to exclusion lists to prevent other applications from injecting into them.[11]

This adversarial relationship with the browser and other endpoint software makes the legacy model unsustainable. It operates by subverting the browser's intended design rather than cooperating with it through official APIs. This inherent conflict is the primary driver for the migration to the modern, SDK-based architecture.

Section 3: Understanding the Modern Method: The Google SDK and Native Messaging

The modern architecture represents a paradigm shift towards a cooperative security model. It leverages official, vendor-supported APIs to achieve its goals, ensuring stability, security, and maintainability.

3.1 The Google Content Analysis Connector SDK

The core of the new solution is the Google Content Analysis Connector SDK, a specialized toolkit for building a bridge between the Chrome browser and a local analysis agent.[3]

Purpose and Scope

This SDK is a C++ library provided by the Chromium project. Its exclusive function is to standardize the communication protocol for content analysis. It is not a general-purpose SDK for building browser extensions or performing other web protection tasks. It provides the necessary components for a third-party DLP vendor to create a native agent that can seamlessly and securely receive content from the browser for inspection.[3]

Key Components

The SDK is composed of several key elements that work in concert:

- **Protocol Buffers (Protobufs)**: The communication contract between the browser and the agent is defined using Google's Protocol Buffers. The .proto files included in the SDK specify the rigid structure of all request and response messages, such as ContentAnalysisRequest and ContentAnalysisResponse. This use of a formal Interface Definition Language (IDL) ensures type safety, provides backward and forward compatibility for the API, and allows for efficient binary serialization of data exchanged between the browser and the agent.[3]
- **Agent-Side Code**: The SDK includes C++ source code that must be compiled and linked into the DLP agent application. This library abstracts away the complexities of the communication channel, handling the low-level tasks of reading messages from standard input, writing responses to standard output, and parsing the Protobuf messages. This allows the DLP vendor to focus on their core competency: the logic for analyzing content for sensitive data.[3]
- **Browser-Side Code**: The SDK also contains the corresponding client-side code, which is already compiled directly into the official Google Chrome and Microsoft Edge browsers. This built-in component is responsible for identifying user actions that require analysis (based on enterprise policies), packaging the relevant content into a Protobuf message, and sending it to the native agent via the Native Messaging channel.[3]

3.2 The Communication Backbone: Chrome Native Messaging

The secure and stable communication between the browser and the DLP agent is made possible by the Chrome Native Messaging API. This API is the official, sanctioned mechanism for a browser to communicate with a separate, locally installed native application, referred to as a "native messaging host".[6]

The Handshake and Registration

Before the browser can communicate with the native host, the host must be registered with the browser. This is a two-step process on Windows that ensures only authorized applications can be launched:

> **Native Messaging Host Manifest**: A JSON manifest file must be created and placed in a specific, protected location on the user's computer. This file acts as a registration card for the native host. It contains critical metadata, including a

unique name for the host (e.g., com.vendor.dlp_agent), the full path to its executable file, and, most importantly, a list of extension IDs or browser origins that are explicitly permitted to connect to it.[7]

Windows Registry Keys: For the browser to discover this manifest file, a specific key must be created in the Windows Registry. The key is created under HKEY_LOCAL_MACHINE\SOFTWARE\Google\Chrome\NativeMessagingHosts\ (for system-wide installation) and must be named identically to the name field in the JSON manifest. The default value of this registry key must contain the full path to the JSON manifest file.[7] This registry entry is the definitive link that allows Chrome to find and validate the native host.

Verifying and managing these registry keys is a central task of this project.

The Communication Protocol

Once registered, the communication follows a simple yet robust protocol:

- The browser launches the native host executable as a separate process. This process isolation is a key security feature.
- All communication occurs over the standard I/O streams: the browser writes messages to the native host's standard input (stdin) and reads responses from the host's standard output (stdout).
- Each message is a JSON payload encoded in UTF-8.
- To handle message framing, each JSON message is preceded by a 32-bit unsigned integer that specifies the exact length of the following message in bytes. The receiving application must first read these four bytes to determine the message size, then read exactly that number of bytes to receive the complete JSON object. This simple length-prefix framing protocol prevents synchronization errors and is a common point of failure if not implemented correctly.[7]

3.3 Architectural Advantages and Comparative Analysis

The modern architecture offers clear and compelling advantages over the legacy model in every significant category, justifying the migration effort.

- **Security**: The Native Messaging model is fundamentally more secure. The native host runs in its own sandboxed process, preventing it from accessing the browser's internal memory or state directly. The manifest file ensures that only explicitly authorized extensions or browser components can initiate communication. This design eliminates the entire class of security risks associated with DLL injection and API hooking.
- **Stability**: Because it relies on a formal, versioned API maintained by the browser vendor, the communication protocol is exceptionally stable across browser updates. The Chromium team is committed to maintaining backward compatibility for these enterprise APIs, which eliminates the fragility and high maintenance cost of the hooking-based approach.
- **Performance**: While all Inter-Process Communication (IPC) introduces some overhead from data serialization and context switching, it is highly predictable and generally negligible for the use cases of content analysis. This is a significant improvement over the unpredictable and often severe performance degradation caused by poorly implemented API hooks or conflicts between multiple injected DLLs.
- **Maintainability**: The architecture promotes a clean separation of concerns. The

browser is responsible for the user interface and event triggering, while the native host is responsible for the analysis logic. This, combined with a well-defined API, makes the entire solution easier to develop, debug, and maintain over its lifecycle.

The following table summarizes the critical differences between the two architectures.

Feature	Legacy Model (DLL Injection & Hooking)	Modern Model (Content Analysis Connector)
Mechanism	Direct process memory injection and API hooking.	Inter-Process Communication (IPC) via stdin/stdout.
Communication Protocol	Undefined, relies on shared memory and function pointers.	Structured, versioned protocol (Native Messaging + Protobufs).
Security Posture	Very High Risk. Mimics malware techniques.[1] Vulnerable to hijacking.[15]	Low Risk. Sanctioned API. Process isolation. Explicit permissions.
Stability	Fragile. Breaks with browser or security software updates. Prone to deadlocks.[11]	Robust. Stable API maintained by Google.
Performance	Unpredictable. Can cause significant slowdowns and conflicts.	Predictable IPC overhead. Generally efficient.
Vendor Support	Unsupported. Operates outside of official browser APIs.	Officially supported and documented by Google/Chromium.[3]
Manageability	Difficult. Requires complex installers and exclusion rules in other security products.	Simple. Managed via standard enterprise policies (GPO/CBCM).

Section 4: Enterprise Deployment and Policy Management

Successfully implementing the new web protection solution requires proficiency in modern enterprise browser management frameworks. The project necessitates skills in both cloud-native and traditional on-premise tools, as browser policies can be configured and enforced through multiple channels.

4.1 Cloud-Native Control: Chrome Browser Cloud Management (CBCM)

Chrome Browser Cloud Management is Google's centralized, web-based console for managing Chrome browsers across an entire organization, regardless of the underlying operating system or whether the devices are joined to a local domain.[16]

Overview and Enrollment

CBCM is a feature of the free Chrome Enterprise Core service. It allows administrators to enforce over 300 policies, manage extensions, and view detailed reports from the Google Admin console.[17] To be managed by CBCM, a browser must first be "enrolled." This process involves generating a unique enrollment token within the Admin console. This token is then deployed to the target machines, typically via a script, GPO, or other endpoint management tool. Once the browser has the token, it registers itself with the organization's CBCM instance and begins pulling down its assigned policies.[19]

Configuring the Content Analysis Connector

The Google Admin console provides specific policies designed to enable and configure the DLP integration. Within the settings for Chrome, under the Chrome Enterprise Connectors section, an administrator can configure policies like Upload content analysis and Bulk text content analysis. For each policy, the administrator selects the DLP vendor from a dropdown list. This selection configures the browser to use the corresponding native messaging host for analysis. This provides a user-friendly, cloud-based method for activating the integration without needing to directly manipulate registry keys on each endpoint.[5]

4.2 On-Premise Control: Windows Group Policy (GPO)

For organizations with a traditional, on-premise Active Directory infrastructure, Windows Group Policy remains the most powerful and common tool for managing user and computer settings on domain-joined Windows devices.

Overview and Administrative Templates

Both Google Chrome and Microsoft Edge provide administrative templates (.admx and .adml files) that integrate directly into the Group Policy Management Editor. These templates expose the browser's configurable settings as standard GPO settings.[20] The first step in managing browsers via GPO is to download the appropriate enterprise bundle (e.g., the Google Chrome Enterprise Bundle) and copy these template files into the domain's central PolicyDefinitions store (typically located in the SYSVOL share). This makes the browser policies available to all administrators for configuration.[20]

Configuring Registry Values and Policies

Your assigned task of setting registry values is a classic GPO function. The Native Messaging Host registration requires specific registry keys to be present for the browser to discover the agent. The DLP agent's installer should ideally create these keys, but it is the administrator's responsibility to understand, verify, and enforce them using tools like Group Policy. The critical keys are:

- HKEY_LOCAL_MACHINE\SOFTWARE\Google\Chrome\NativeMessagingHosts\{host_na

me}
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Edge\NativeMessagingHosts\{host_name}

The default value for each of these keys must be a string (REG_SZ) containing the full path to the native host's JSON manifest file.[7]

Beyond registry settings, GPO is used to enforce other critical policies. For example, an administrator can use the Configure the list of force-installed apps and extensions policy (found under Administrative Templates > Google > Google Chrome > Extensions) to silently deploy a required companion extension to all users in an organizational unit.[23] Microsoft Edge offers a parallel set of policies for the same purpose.[21]

4.3 Comparative Analysis of Management Frameworks

The choice between CBCM and GPO is not always mutually exclusive; many organizations use a hybrid approach. Understanding the strengths of each is key to effective management.

- **Chrome Browser Cloud Management (CBCM)**: Its primary strength is its cross-platform and location-agnostic nature. It is the ideal solution for managing browsers in heterogeneous environments (Windows, macOS, Linux) and for supporting devices that are not consistently connected to the corporate network or are not joined to the Active Directory domain. CBCM also provides powerful, built-in reporting on browser versions, installed extensions, and policy compliance across the entire fleet.[17]
- **Windows Group Policy (GPO)**: GPO is the standard for tightly controlled, on-premise Windows environments. It offers deep, granular control over the entire operating system, not just the browser. For domain-joined machines, policy application through GPO is often faster and more robust.
- **Hybrid Strategy**: A common and effective strategy is to use both tools. GPO can be used for the initial bootstrap process on domain-joined Windows machines, such as deploying the CBCM enrollment token. Once enrolled, the more flexible CBCM can be used for the ongoing, day-to-day management of browser-specific policies. This approach leverages the strengths of both systems.

The requirements of this project—encompassing both the configuration of on-premise registry keys and the use of a cloud-based management console—mandate a hybrid IT administration skillset. Proficiency in both traditional GPO management and modern cloud-native tools like CBCM is necessary to successfully deploy, manage, and troubleshoot the solution across the diverse range of devices found in a modern enterprise.

Section 5: Foundational Learning Path and Strategic Recommendations

This final section synthesizes the preceding analysis into an actionable learning plan and provides key strategic recommendations for the successful implementation of the project.

5.1 Core Competencies for Project Success

To excel in this project, a deep understanding of several interconnected technology domains is required. Mastery of these areas will enable effective implementation, troubleshooting, and long-term maintenance of the web protection solution.

- **Windows Internals**: A strong foundation in the architecture of the Windows operating system is crucial. This includes a deep understanding of the Windows Registry's structure and purpose, the concepts of processes and threads, and familiarity with the core Win32 API. This knowledge is essential for debugging the Native Messaging host and fully comprehending the risks of the legacy DLL injection method.
- **Enterprise IT Administration**: Practical, hands-on skills in using the Group Policy Management Editor are non-negotiable. This includes creating, linking, and troubleshooting GPOs, as well as deploying administrative (ADMX) templates. Concurrently, familiarity with cloud-based administration consoles, specifically the Google Admin console for CBCM, is required to manage the entire device fleet.
- **Networking and Protocols**: A conceptual understanding of Inter-Process Communication (IPC), the role of standard I/O streams (stdin/stdout), and the principles of data serialization are important. Specific knowledge of formats like JSON and a high-level understanding of Protocol Buffers will aid in troubleshooting communication issues between the browser and the native agent.
- **Browser Technology**: A working knowledge of the modern browser architecture is necessary. This includes understanding the role and structure of the manifest.json file for extensions and the browser's security model, which underpins policies like the Content Security Policy (CSP).[25]
- **Security Principles**: A firm grasp of fundamental security concepts is paramount. This includes process isolation, sandboxing, the principle of least privilege, and a clear understanding of the threats posed by techniques like API hooking and code injection.

5.2 Recommended Learning Trajectory

A structured, hands-on learning approach will be the most effective way to build the required competencies. The following sequence is recommended:

Master Native Messaging First: This is the technological core of the new architecture. The most valuable learning exercise is to build a simple "hello world" native messaging host using a familiar language (e.g., Python, C#, or even a simple batch script) and a companion browser extension that communicates with it. Manually create the required JSON manifest file and the Windows Registry keys to establish the connection. Successfully passing a message from the extension to the native host and back will solidify the understanding of the entire pipeline, from registration to communication protocol.[7]

Explore GPO for Browser Management: In a virtualized lab environment, set up a test Active Directory domain controller and a domain-joined Windows client. Download the Google Chrome and Microsoft Edge administrative templates and deploy them to the domain's PolicyDefinitions store. Practice creating and applying

GPOs that configure various browser settings. Specifically, practice using the ExtensionInstallForcelist policy to automatically deploy an extension from the Chrome Web Store. Use the chrome://policy and edge://policy internal pages on the client machine to verify that the GPO settings are being correctly received and applied.[22]

Dive into Chrome Browser Cloud Management: Sign up for a free Chrome Enterprise Core trial account. Within the Google Admin console, practice generating an enrollment token. On a test machine (which does not need to be domain-joined), deploy this token to enroll the browser. Explore the policy interface within the console and locate the Chrome Enterprise Connectors settings. Practice configuring these settings to simulate enabling the DLP integration.[5]

Review the Legacy Concepts: Once a solid understanding of the modern, sanctioned architecture is established, revisit the concepts of DLL injection and API hooking. Study the Win32 API calls involved (CreateRemoteThread, etc.) and the mechanics of IAT hooking. This contextual knowledge is invaluable for articulating the technical and business benefits of the migration to other stakeholders and for understanding potential legacy conflicts during the transition period.[1]

5.3 Key Implementation Considerations

Beyond the technical learning, several strategic considerations will be critical to the project's success.

- **Collaboration with the Security Team**: The legacy solution's history of creating conflicts with endpoint security software makes the Information Security team a key stakeholder. Proactive engagement is essential. Clearly explain the new architecture, emphasizing the move from malware-like injection techniques to a secure, vendor-supported API. This will build confidence and ensure a smooth rollout without triggering false alarms from EDR or antivirus systems.
- **Pilot Testing**: Before a full enterprise-wide deployment, a thorough pilot phase must be conducted. This pilot should include a representative sample of users from different departments and, crucially, devices with the full standard corporate software stack installed. The goal is to verify that the new DLP agent and its communication with the browser are stable, performant, and do not conflict with any other business-critical or security applications.
- **Installer and Deployment Strategy**: Close collaboration with the DLP vendor and the internal software packaging team is required. The agent's installer must be robust, correctly creating the Native Messaging Host manifest files and the associated registry keys for all supported browsers (Chrome, Edge, etc.) on all targeted versions of Windows.
- **User Communication and Training**: While the new solution should be largely transparent to end-users, there may be subtle changes in their experience. For example, a file upload may have a brief delay while the content is scanned by the DLP agent. Clear, concise communication should be prepared to inform users about the new data protection measures, explaining the purpose of these controls and what to expect.

While checking in Chrome enterprise -> Set registry values
Unchecking it then -> Remove registry values

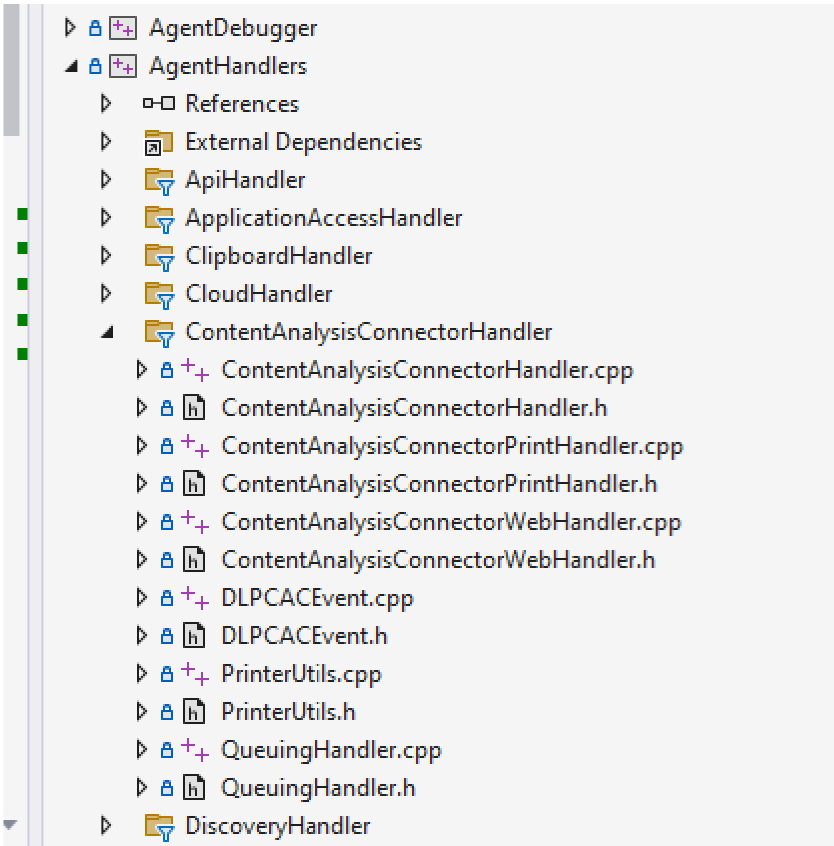
Here are the ultra detailed notes from the meeting:

- **Main Topic:** The discussion centered around enabling Web protection roles, specifically for Firefox, by setting registry values instead of DLL injection, using an SDK integration approach.
- **Key Decisions/Outcomes:**
    - The team will use SDK integration for Web protection roles in browsers like Chrome, Firefox, and Edge, to avoid DLL injection.
    - Registry entries in Windows will be used to configure browser policies to allow DLP to access data.
    - A policies.json file for Firefox will be copied into the Firefox distribution folder to enable these policies.
    - Specific registry keys under HQ Local Machines\Software\Policies\Mozilla\Firefox will be set, including enable plaintext for clipboard interception.
    - If the Chrome enterprise or Firefox option is unchecked in Wcc configuration, the corresponding registry keys should be deleted.
- **Unresolved Issues:**
    - Siva Sankari Ganesan is facing an issue with EPO extensions not getting the policy, preventing them from fully testing the implementation.
    - Adarsh Anand needs Ganesha to provide the specific registry values to change.
- **Action Items:**
    - Adarsh Anand:
        - Ask Ganesha for the specific registry values to change.
        - Go through the shared documents and code related to registry entries and how they are used.
    - Siva Sankari Ganesan:
        - Share the document and conference page related to Firefox policies and SDK integration.
        - Continue testing the implementation, specifically checking for the creation of registry entries after pulling the policy.

From <https://meet.google.com/fgd-tpip-rnp>

How do I get access to Google Chrome admin console?
How do I test them - should I be creating admin console CBCM and pushing policies to endpoints?

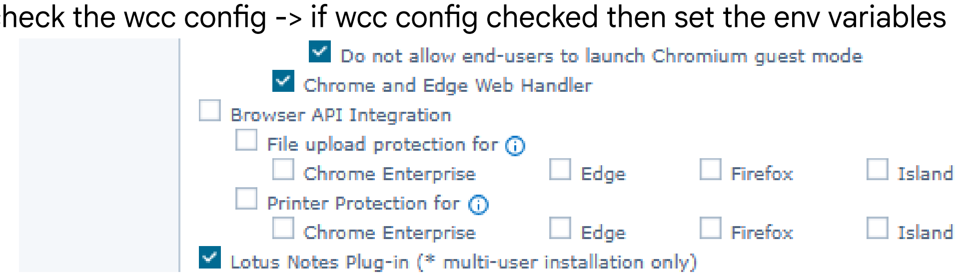
Files to work with -
AgentHandler > ContentAnalysisConnectorHandler



Manually check registry values

Wheter google sdk works or not? -> disabling dll and enable gsdk
Relaunch chrome after making changes -> web protection

========== WORK ==============

1. Uncheck wcc config values + Check if native web protection with dll injection working -> OK
2. Now check the wcc config -> if wcc config checked then set the env variables
    1. 

3. Make code changes of env vars -> once we have the code changes done -> OK
    1. Do a local signing and replace the 10 fcag binaries on the test vm

4. Acceptance criteria
    1. Check if registry entries are getting reflected
    2. Check if web blocking is working when browser api integration is checked
    3. Check if the management shows the DLP extension

Working with the latest builds - 363
https://artifactory.trellix.com/ui/repos/tree/General/DLPE_AGENT-local/DLPe_Agent_11.12.1/b363/TAG_AGENT_11.12.1.363/Signed_Package/HDLP_Agent_Package.zip

DLP Management Extension
https://artifactory.trellix.com/ui/repos/tree/General/DLP_Extension-local/UDLP_Extension/11.13.0/11.13.0.6/b6/Signed/DLP_Mgmt_Extension.zip

```
// Chrome registry settings when CAC enabled
const std::wstring chromePolicyKey =
L"HKEY_LOCAL_MACHINE\\SOFTWARE\\Policies\\Google\\Chrome";
const std::wstring enrollmentTokenValueName = L"CloudManagementEnrollmentToken";
const std::wstring enrollmentTokenValue = L"93628663-f462-4487-bdf8-e232fd9e1e33";
const std::wstring fullTokenPath = chromePolicyKey + L"\\" + enrollmentTokenValueName;

std::cout << "mIsCacEnabledForWebOnChrome value is -> " <<
mIsCacEnabledForWebOnChrome << std::endl;

if (mIsCacEnabledForWebOnChrome)
{
	ODEBUG_CAC_HANDLER1("Web CAC for Chrome is enabled. Setting enrollment token.");

	// Create the key path and set the string value.
	// The underlying RegistrySetStringValue will create the key if it doesn't exist.
	int result = RegistryUtils::RegistrySetStringValue(fullTokenPath, enrollmentTokenValue);
	if (result != ERROR_SUCCESS)
	{
		OERROR_CAC_HANDLER2("Failed to set Chrome enrollment token. Error: %d",
		result);
	}
}
else
{
	ODEBUG_CAC_HANDLER1("Web CAC for Chrome is disabled. Removing enrollment
	token if it exists.");

	// Delete the value to clean up the configuration.
	int result = RegistryUtils::RegistryDeleteValue(fullTokenPath);

	// It's not an error if the value wasn't there to begin with.
	if (result != ERROR_SUCCESS && result != ERROR_FILE_NOT_FOUND)
	{
		OERROR_CAC_HANDLER2("Failed to delete Chrome enrollment token. Error: %d",
```

```
            result);
        }
}
```

-

Following up on our call yesterday with the requested items from our end. It sounds like we are on track for the Trellix connector reaching GA **the first week of December.** See below for the requested resources:

From <https://jira.trellix.com/secure/RapidBoard.jspa?rapidView=3962&projectKey=DLPW&view=planning&selectedIssue=DLPW-19019&quickFilter=217947&issueLimit=100#>

TODO

  - Move registry entries in Agent/Dlp/AgentHandlers/ContentAnalysisRegistryEntries.h after merge of DLPW-19061: Implement Registry Entries for Firefox. #1439
  - Checking edge versions only do registry changes

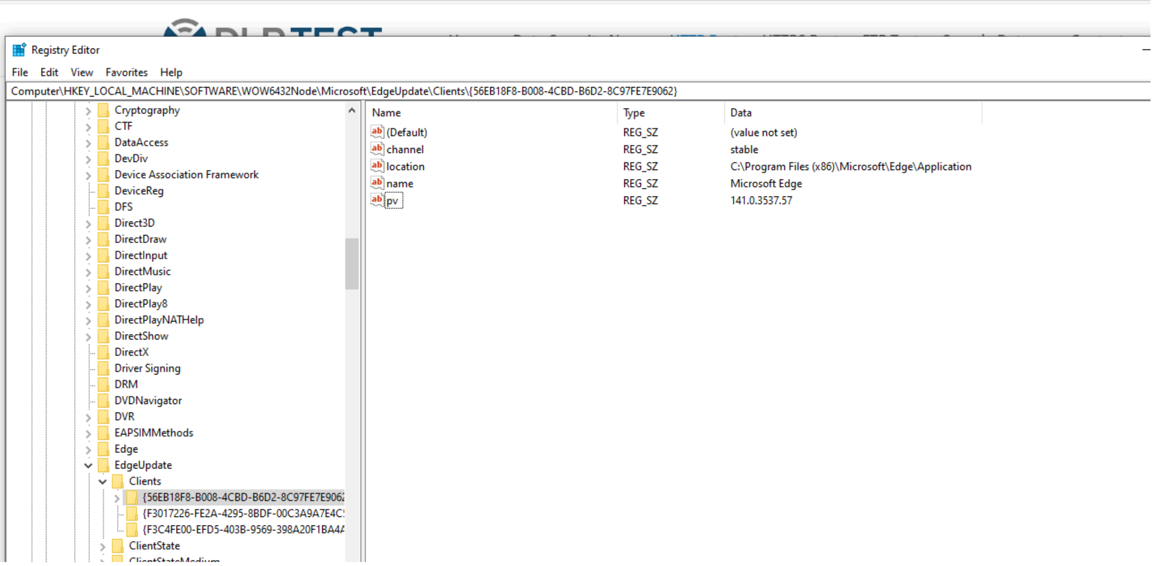
Checking Edge versions

Under HKLM\SOFTWARE\WOW6432Node\Microsoft\EdgeUpdate\Clients{56EB18F8-B008-4CBD-B6D2-8C97FE7E9062} there are two possible interesting values:

If only 'pv' is present, this is the version of Edge Stable that is installed and will run when launched.

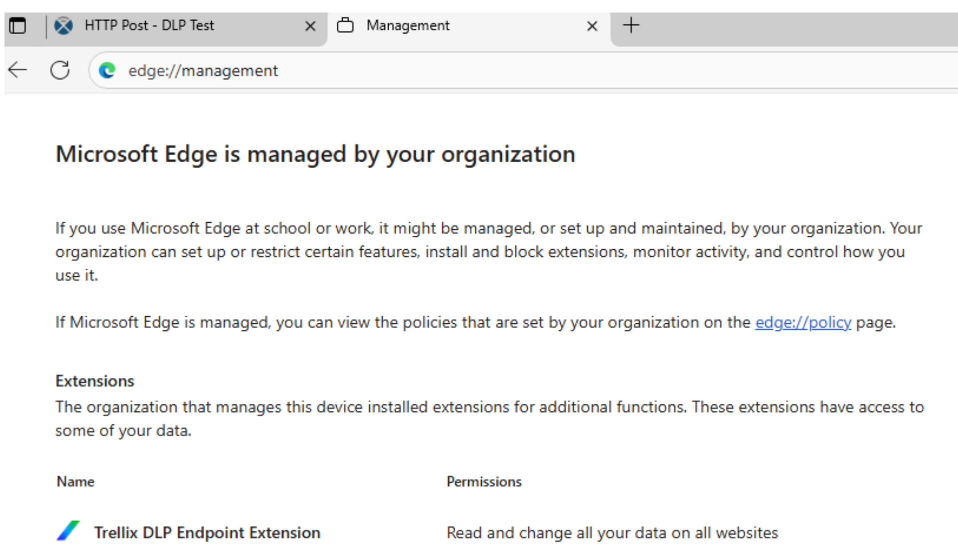
If both 'pv' and 'opv' are present, then 'opv' is the currently running version of Edge and 'pv' is the staged update of Edge. When Edge shuts down, the version

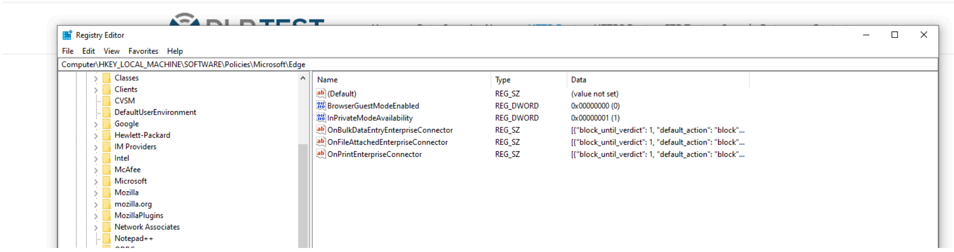
swap will happen and 'opv' is delete



d and 'pv' will be the only remaining value

Tested and works fine





From <https://github.trellix.com/trellix-products/DLPe.Agent/pull/1442>

Here are the detailed notes from the meeting:

- **Problem Statement:** Adarsh Anand's recent changes are not effectively deleting registry entries and tokens after project uninstallation, despite the logical correctness of the changes.
- **Initial Investigation:**
  - Mallikharjuna Yallavula advised Adarsh to review the `uninstall customer` function in the code and its calling locations.
  - They discussed the installer's role in calling this function and the potential for the uninstallation process to use an outdated or incorrect DLL.
- **Logging Issue:**
  - A key challenge identified was the absence of installation and uninstallation logs in the McAfee logs folder, hindering debugging efforts.
- **Proposed Solution - Build Generation:**
  - Mallikharjuna suggested generating a new build from Adarsh's private branch using Jenkins (Orbit jobs).
  - This build process is estimated to take 4-4.5 hours.
  - The new build will help validate Adarsh's changes and confirm that the correct DLLs are being utilized during uninstallation.
- **Further Debugging Steps:**
  - Mallikharjuna explained that the installer might be retaining temporary files or copies of DLLs in encrypted locations, which could be used during uninstallation instead of the updated DLLs.
  - They discussed searching for `driver install.dll` on the C drive to identify all instances of the DLL.
- **Understanding the Advanced Installer (AIP) File:**
  - Mallikharjuna clarified that the AIP file is used by Advanced Installer, a product that creates installer packages (EXE files) for product deployment.
  - The AIP file defines deployment steps, prerequisites, DLL placement, and custom actions like driver installation/updates.
  - When an EXE generated from an AIP file is run, it applies the required product files. When deployed through EPO, these EXE files are bundled into a single

EPO package.

- **Monitoring Installation Process:**
    - Mallikharjuna demonstrated how to monitor the installation process using the agent monitor on the endpoint, observing tasks like "product installed become active" and "enforcing policies task is freed" to confirm completion.
- **DLL Usage Discrepancy:**
    - It was noted that while Adarsh's DLL is in the system, it doesn't appear to be used during uninstallation, suggesting the installer is using a different, possibly temporary, DLL.
- **Manual Installation and Logging:**
    - Mallikharjuna explained how to manually install the product using the EXE file from the build location (Orbit artifacts) via the command prompt, which allows for the generation of a log file.
    - Uninstallation, however, should be done through the Windows Control Panel to ensure log generation.
- **Next Steps for Adarsh:**
    - Continue testing with the newly generated build from Orbit.
    - If the issue persists, consider writing a simple tool to call the DLL code directly to verify its execution. Mallikharjuna offered to help create this tool if needed and suggested using Copilot or Gemini for code assistance.

ContentAnalysis

Configure Firefox to use an agent for Data Loss Prevention (DLP) that is compatible with the Google Chrome Content Analysis Connector Agent SDK.

`AgentName` is the name of the DLP agent. This is used in dialogs and notifications about DLP operations. The default is “A DLP Agent”.

`AgentTimeout` is the timeout in number of seconds after a DLP request is sent to the agent. After this timeout, the request will be denied unless `TimeoutResult` is set to 1 or 2. The default is 300.

`AllowUrlRegexList` is a space-separated list of regular expressions that indicates URLs for which DLP operations will always be allowed without consulting the agent. The default is “^about:(?!blank|srcdoc).*”, meaning that any pages that start with “about:” will be exempt from DLP except for “about:blank” and “about:srcdoc”, as these can be controlled by web content.

`BypassForSameTabOperations` indicates whether Firefox will automatically allow DLP requests whose data comes from the same tab and frame - for example, if data is copied to the clipboard and then pasted on the same page. The default is false.

`ClientSignature` indicates the required signature of the DLP agent connected to the pipe. If this is a non-empty string and the DLP agent does not have a signature with a Subject Name that exactly matches this value, Firefox will not connect to the pipe. The default is the empty string.

`DefaultResult` indicates the desired behavior for DLP requests if there is a problem connecting to the DLP agent. The default is 0.

Value	Description
0	Deny the request (default)
1	Warn the user and allow them to choose whether to allow or deny
2	Allow the request

`DenyUrlRegexList` is a space-separated list of regular expressions that indicates URLs for which DLP operations will always be denied without consulting the agent. The default is the empty string.

`Enabled` indicates whether Firefox should use DLP. Note that if this value is true and no DLP agent is running, all DLP requests will be denied unless `DefaultResult` is set to 1 or 2.

`InterceptionPoints` controls settings for specific interception points.

- The `Clipboard` entry controls clipboard operations for files and text.
    - `Enabled` indicates whether clipboard operations should use DLP. The default is true.
    - `PlainTextOnly` indicates whether to only analyze the text/plain format on the clipboard. If this value is false, all formats will be analyzed, which some DLP agents may not expect. Regardless of this value, files will be analyzed as usual. The default is true.
- The `Download` entry controls download operations. (Added in Firefox 142, Firefox ESR 140.2)
    - `Enabled` indicates whether download operations should use DLP. The default is false.
- The `DragAndDrop` entry controls drag and drop operations for files and text.
    - `Enabled` indicates whether drag and drop operations should use DLP. The default is true.
    - `PlainTextOnly` indicates whether to only analyze the text/plain format in what is being dropped. If this value is false, all formats will be analyzed, which some DLP agents may not expect. Regardless of this value, files will be analyzed as usual. The default is true.
- The `FileUpload` entry controls file upload operations for files chosen from the file picker.

    - `Enabled` indicates whether file upload operations should use DLP. The default is true.
- The `Print` entry controls print operation.
    - `Enabled` indicates whether print operations should use DLP. The default is true.

`IsPerUser` indicates whether the pipe the DLP agent has created is per-user or per-system. The default is true, meaning per-user.

`PipePathName` is the name of the pipe the DLP agent has created and Firefox will connect to. The default is “path_user”.

`ShowBlockedResult` indicates whether Firefox should show a notification when a DLP request is denied. The default is true.

`TimeoutResult` indicates the desired behavior for DLP requests if the DLP agent does not respond to a request in less than `AgentTimeout` seconds. The default is 0.

https://jira.trellix.com/browse/DLPW-19084

https://jira.trellix.com/browse/DLPW-19022

https://jira.trellix.com/browse/DLPW-19019

Reverting Registry Creation (Cleanup Logic)

The following logic is crucial for ensuring that when the **Google SDK** option (Content Analysis Connector - CAC) is disabled in the ePO GUI, all corresponding registry keys and policies are cleanly removed from the endpoint system. This ensures a clean fallback to the legacy method or simply a deactivation of the protection mechanism.

This logic is directly implemented in the `AgentHandler > ContentAnalysisConnectorHandler` and is executed when a policy pull indicates that `mIsCacEnabledForWebOnChrome` or the corresponding Edge/Firefox flags are set to false.

1. Chrome Registry Reversion

When the Google SDK option for Chrome is unchecked in the ePO GUI, the following action must be executed:

Action	Registry Path	Value Name
Delete Value	`HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome`	`CloudManagementEnrollmentToken`

The recent investigation highlighted that the registry cleanup logic must be correctly executed during the product's **uninstallation** process, not just during policy unchecking. Adarsh needs to ensure his code changes for deleting these registry entries are correctly called by the `uninstall customer` function and that the installer is using the **latest, updated DLL** for cleanup.

https://jira.trellix.com/browse/DLPW-19895

Chrome config

Windows Registry Editor Version 5.00

; Set up the enrollment token for organizational unit: Trellix
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome]
"CloudManagementEnrollmentToken"="93628663-f462-4487-bdf8-e232fd9e1e33"

Edge config

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Edge]
"OnBulkDataEntryEnterpriseConnector"="[{\"block_until_verdict\": 1, \"default_action\": \"block\", \"enable\":[{\"tags\":[\"dlp\"],\"url_list\":[\"*\"]}], \"minimum_data_size\": 1, \"service_provider\":\"trellix\"}]"
"OnFileAttachedEnterpriseConnector"="[{\"block_until_verdict\": 1, \"default_action\": \"block\", \"enable\":[{\"tags\":[\"dlp\"],\"url_list\":[\"*\"]}], \"service_provider\":\"trellix\"}]"
"OnPrintEnterpriseConnector"="[{\"block_until_verdict\": 1, \"default_action\": \"block\", \"enable\":[{\"tags\":[\"dlp\"],\"url_list\":[\"*\"]}], \"service_provider\":\"trellix\"}]"

Firefox config

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis]
"AgentName"="Trellix DLP Agent"
"AllowUrlRegexList"=""
"DenyUrlRegexList"=""

"PipePathName"="Trellix_DLP"
"AgentTimeout"=dword:0000003c
"BypassForSameTabOperations"=dword:00000001
"DefaultResult"=dword:00000002
"Enabled"=dword:00000001
"IsPerUser"=dword:00000001
"TimeoutResult"=dword:00000000
"ShowBlockedResult"=dword:00000001
"ClientSignature"="Musarubra US LLC"

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Clipboard]
"Enabled"=dword:00000001
"PlainTextOnly"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Downloads]
"Enabled"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\DragAndDrop]
"Enabled"=dword:00000001
"PlainTextOnly"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\FileUpload]
"Enabled"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Print]
"Enabled"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\FileUpload]
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Print]

Policies.json file

GSDK UT

Oct 17, 2025

DLPW UT understanding and development

Invited Mallikharjuna Yallavula Siva Sankari Ganesan GANPAT JOSHI Swatesh Ambast Adarsh Anand Ganesh A Rao

Attachments DLPW UT understanding and development

Meeting records Transcript

DLPW UT understanding and development - 2025/10/17 15:58 GMT+05:30 -...

Summary

Mallikharjuna Yallavula provided an overview of DLP unit test development, explaining the framework's use of CPP unit test for both unit and integration tests within the sanity test project. Adarsh Anand inquired about distinguishing test types and automating actions like file uploads, which Mallikharjuna Yallavula confirmed as feasible but noted the lack of automation for browser uploads. Key talking points included the distinction between unit and integration tests, the sanity test project's structure, test case development and execution, and the current challenges with test coverage in orbit, particularly regarding browser upload automation.

Details

Notes Length: Standard

- **DLP Unit Test Development Overview** Mallikharjuna Yallavula initiated the meeting to discuss DLP unit test development, focusing on understanding the existing framework and outlining future development procedures for the DLP product. They planned to cover slides, conduct a code walkthrough, and hold a Q&A session (00:00:00).

- **Unit Test Goals and Implementation** Mallikharjuna Yallavula explained that unit tests are for validating a specific piece of code, independent of the overall product, to ensure it works for all inputs and handles exceptions (00:01:13). They highlighted that DLP's unit test framework is built using CPP unit test, a third-party library, and is part of the sanity test project (00:03:30).
- **Integration Test Overview** Mallikharjuna Yallavula described integration tests as feature validations that confirm if customer-expected features function correctly, typically involving product installation and dependency components (00:04:34). They noted that DLP's integration test framework is also part of the sanity test project and utilizes the same CPP unit test library as the unit tests (00:05:45).
- **Distinguishing Unit and Integration Tests** Adarsh Anand inquired about distinguishing between unit and integration tests by examining function code (00:19:01). Mallikharjuna Yallavula clarified that unit tests involve direct function calls without product-related parameter or configuration settings (00:20:06), while integration tests, though sometimes directly calling product-related functions, often necessitate product installation and policy settings to validate features (00:25:09) (00:27:21).
- **Sanity Test Project Structure** Mallikharjuna Yallavula detailed the sanity test project, explaining that both unit tests and integration tests use it. They outlined that the project includes necessary libraries like OpenSSL and product-related MDK libraries for successful compilation, similar to those used by DLP processes, ensuring the same code can be validated within the sanity test (00:10:05).
- **Test Case Development and Registration** Mallikharjuna Yallavula explained that when building a test case, a class derived from `I Sanity Test` needs to be created, along with specifying a test suite and a short name for registration. This registration process allows the CPP unit test framework to manage and invoke the correct test cases at runtime based on their short names (00:15:05).

- **Executing Sanity Tests** Mallikharjuna Yallavula demonstrated how to execute `sanity.exe`, explaining that it can take command-line arguments to run specific test cases by their short names (00:32:14). They also mentioned that if no arguments are provided, all registered test cases will be executed, and the execution time is recorded (00:33:49).
- **Test Case Execution in Orbit Pipeline** Mallikharjuna Yallavula described the `run_sanity.bat` batch file used by the Jenkins pipeline to execute tests after product build (00:37:24). They noted that this script handles product installation and directs test results and input files to specified directories (00:38:37).
- **Current Test Case Coverage and Challenges** Mallikharjuna Yallavula acknowledged that while many test cases are built, not all are currently running successfully in orbit due to environmental issues like missing network or USB drives, leading to the disablement of some tests (00:36:26) (00:40:48). They emphasized the need to reactivate these failing tests and to add new unit and integration tests for all new features to improve coverage (00:30:57) (00:41:49).
- **Automating Actions within Tests** Adarsh Anand inquired about the possibility of automating actions like file uploads and email sending via C++ code for testing (00:52:56). Mallikharjuna Yallavula confirmed its feasibility, noting that their automation team already possesses such capabilities and that tools like Microsoft's automation framework can be used to interact with browser controls (00:54:03).
- **Browser Upload Automation** Mallikharjuna Yallavula highlighted the need for automation in browser upload functionality to identify issues with new changes at the unit test level. Adarsh Anand confirmed the current lack of automation for browser uploads in the existing code. Mallikharjuna Yallavula suggested building a new test framework for web production rules using GSTK if an existing solution is not found, emphasizing the opportunity to develop an entire suite for this purpose (00:59:16).

Suggested next steps

- ☐ The group will add the recent short names, including ADB and ID related names, to the unit test labels.
- ☐ Adarsh Anand will cross-check with the automation team or research how to launch a browser and upload a file to it using C++ code.
- ☐ Mallikharjuna Yallavula will walk through the existing code to check if any test framework is already defined or implemented, and if not, plan and implement a new test framework for web production rule using GSTK.
- ☐ Mallikharjuna Yallavula will spend some time to find automation things implemented correctly.

Remove all the unnecessary code that you have written and create a very basic minimal function the main goal of the project is to do whatever testing that we have done on the functions that we have created and don't create any extra other functions on your own we need to first perform the unit test on all of the functions that we have created don't do explicit or registry settings or any changes something just call the function and check if everything is working fine

Refine the code by removing all unnecessary elements, focusing on a minimal, basic function. The primary goal is to conduct unit tests solely on the existing functions. Do not introduce any new functions. The testing should involve direct function calls to verify correct operation, avoiding any explicit changes to registry settings or system configurations.

Task

Goal: Develop Comprehensive Unit Tests for Google Content Analysis Module

1. **Core Development:**
    - Create all necessary files and ensure the code for the Google content analysis module is fully functional.

2. **Registry Validation Function (Critical Requirement):**
    - Develop a dedicated function to validate that all required registries are correctly inserted into the Windows environment during any registry-related changes.
    - This function must confirm that no corresponding registries are inadvertently omitted.
3. **Testing Sequence:**
    - Once the registry validation function is operational, perform a complete sanity test of the project.
    - Subsequently, proceed with the development of additional unit testing functions.

Reference the existing cppunit test project structure for foundational knowledge.

Now we need to come up with a framework that whenever we test one function we have to perform the file upload and make sure it is blocking based on keyword

Visual Studio Project File (.vcxproj) Crash Course

A `.vcxproj` file is an XML-based project file used by Visual Studio and MSBuild to define how to compile and link a C++ project. Here's a breakdown:

File Structure Overview

XML
```xml
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Build" ToolsVersion="14.0"
xmlns="...">
  <!-- Project configuration definitions -->
  <!-- Global properties -->
  <!-- Per-configuration settings -->
  <!-- Source files to compile -->
  <!-- Import MSBuild targets -->
</Project>
```

Key Tags and Sections

1. `<ProjectConfiguration>` - Platform/Configuration Combinations

```xml
<ItemGroup Label="ProjectConfigurations">
  <ProjectConfiguration Include="Release|x64">
    <Configuration>Release</Configuration>
    <Platform>x64</Platform>
  </ProjectConfiguration>
</ItemGroup>
```

- Defines all build configurations (Release/Debug) × platforms (Win32/x64/ARM64/ARM64EC)
- Each combination gets its own build settings

2. `<PropertyGroup>` - Global and Per-Configuration Properties

```xml
<PropertyGroup
Condition="'$(Configuration)|$(Platform)'=='Release|x64
'">
  <TargetName>sanity</TargetName>
  <OutDir>..\..\$(BIN_OUT)</OutDir>
  <LinkIncremental>false</LinkIncremental>
</PropertyGroup>
```

- **`Condition`**: Applies settings only when specific configuration/platform is active
- **`TargetName`**: Output executable name (sanity.exe)
- **`OutDir`**: Where compiled binary is placed
- **`LinkIncremental`**: Enable/disable incremental linking (faster rebuilds)

3. `<ItemDefinitionGroup>` - Compiler and Linker Settings

This is where most configuration happens. Each configuration/platform gets its own group.

`<ClCompile>` - Compiler Settings

```xml
<ClCompile>

<PreprocessorDefinitions>_CONSOLE;U_STATIC_IMPLEMENTATI
ON</PreprocessorDefinitions>
  <AdditionalIncludeDirectories>
    $(OPENSSL_INC);
    ..\..\..\OPG\McAfeeCommon;
    $(ACE_INC);
    %(AdditionalIncludeDirectories)
  </AdditionalIncludeDirectories>
</ClCompile>
```

Key Compiler Tags:

- **`<PreprocessorDefinitions>`**: Define macros (like `-D` flag)
  - Example: `_CONSOLE` defines `#define _CONSOLE`
  - `;` separates multiple definitions
- **`<AdditionalIncludeDirectories>`**: Header search paths (like `-I` flag)
  - Environment variables: `$(VARIABLE_NAME)`
  - Relative paths: `..\..\..\path`
  - `%(AdditionalIncludeDirectories)` = inherit from parent/defaults
- **Other Common Tags:**
  - `<WarningLevel>`: W0-W4 warning levels
  - `<Optimization>`: Disabled, MaxSpeed, MinSize, Full
  - `<RuntimeLibrary>`: MultiThreaded, MultiThreadedDLL, etc.
  - `<LanguageStandard>`: stdcpp14, stdcpp17, stdcpp20, etc.

`<Link>` - Linker Settings

```xml
<Link>
  <SubSystem>Console</SubSystem>
  <AdditionalLibraryDirectories>
    $(ZLIB_LIB);$(ACE_LIB);$(OPENSSL_LIB_X64_DLL)
  </AdditionalLibraryDirectories>
  <AdditionalDependencies>
    kernel32.lib;user32.lib;libssl.lib;cac_agent.lib
```

```
  </AdditionalDependencies>
  <IgnoreSpecificDefaultLibraries>
    legacy_stdio_definitions.lib
  </IgnoreSpecificDefaultLibraries>
</Link>
```

Key Linker Tags:

- **`<SubSystem>`**: Console, Windows, Native
    - Console = command-line app with main()
    - Windows = GUI app with WinMain()
- **`<AdditionalLibraryDirectories>`**: Where to search for .lib files (like `-L` flag)
    - Paths separated by `;`
- **`<AdditionalDependencies>`**: Which .lib files to link (like `-l` flag)
    - System libraries: `kernel32.lib`, `user32.lib`
    - Third-party libraries: `cac_agent.lib`, `libprotobuf.lib`
    - `%(AdditionalDependencies)` inherits defaults
- **`<IgnoreSpecificDefaultLibraries>`**: Exclude specific libraries
    - Useful when you have symbol conflicts (our `legacy_stdio_definitions.lib` fix)
    - Alternative to `/NODEFAULTLIB:library.lib`
- **`<DelayLoadDLLs>`**: Load DLLs lazily at runtime instead of startup
- **`<OptimizeReferences>`**: true = remove unused functions/data (/OPT:REF)
- **`<EnableCOMDATFolding>`**: true = merge identical functions (/OPT:ICF)
- **`<TargetMachine>`**: MachineX86, MachineX64, MachineARM64, MachineARM64X

4. `<ItemGroup>` - Source Files

XML

```xml
<ItemGroup>
  <ClCompile
Include="Tests\TestContentAnalysisRegistry.cpp" />
  <ClCompile Include="Tests\TestClassification.cpp" />
</ItemGroup>

<ItemGroup>
```

```xml
  <ClInclude
Include="Tests\TestContentAnalysisRegistry.h" />
</ItemGroup>
```

- **`<ClCompile>`**: .cpp files to compile
- **`<ClInclude>`**: .h header files (for IDE organization, not compiled directly)
- **`<ResourceCompile>`**: .rc resource files

5. `<Import>` - Import MSBuild Targets

```xml
<Import
Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props"
/>
<Import
Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
```

- Brings in standard build rules from Visual Studio
- Usually at beginning and end of file

Environment Variables and Macros

MSBuild uses `$()` syntax for variables:

```xml
$(CAC_AGENT_INC)      <!-- Custom environment variable
-->
$(Configuration)       <!-- Current config:
Release/Debug -->
$(Platform)            <!-- Current platform:
x64/Win32/ARM64 -->
$(ProjectDir)          <!-- Project directory -->
$(OutDir)              <!-- Output directory -->
$(VCTargetsPath)       <!-- VS installation path -->
```

Property inheritance:

```xml
%(AdditionalIncludeDirectories)  <!-- Inherit from
parent settings -->
```

How It Works - Build Process

1. **Configuration Selection**: User selects Release|x64
2. **Property Evaluation**: MSBuild evaluates all PropertyGroups with matching Condition
3. **Compiler Phase**:
   - Processes each `<ClCompile>` file
   - Uses settings from `<ItemDefinitionGroup Condition="Release|x64"><ClCompile>`
   - Searches include paths, applies preprocessor definitions
   - Produces .obj files
4. **Linker Phase**:
   - Uses settings from `<Link>` section
   - Searches library directories
   - Links all .obj files + specified .lib files
   - Produces final .exe/.dll
5. **Output**: Binary written to `$(OutDir)\$(TargetName).exe`

Common Patterns in Your File

Multi-Platform Support

```xml
<!-- Win32 uses 32-bit libraries -->
<AdditionalLibraryDirectories>$(CAC_AGENT_LIB_32)</AdditionalLibraryDirectories>

<!-- x64 uses 64-bit libraries -->
<AdditionalLibraryDirectories>$(CAC_AGENT_LIB)</AdditionalLibraryDirectories>

<!-- ARM64 uses ARM libraries -->
<AdditionalLibraryDirectories>$(CAC_AGENT_LIB_ARM64)</AdditionalLibraryDirectories>
```

Conditional Compilation

```xml
<PreprocessorDefinitions>_CONSOLE;U_STATIC_IMPLEMENTATI
```

```
ON;%(PreprocessorDefinitions)</PreprocessorDefinitions>
```

- Defines _CONSOLE macro in all .cpp files
- Code can use `#ifdef _CONSOLE` to conditionally compile

Library Dependencies

XML

```xml
<AdditionalDependencies>
  kernel32.lib;         <!-- Windows API -->
  libssl.lib;           <!-- OpenSSL -->
  cac_agent.lib;        <!-- Content Analysis SDK -->
  libprotobuf.lib;      <!-- Protocol Buffers -->
  %(AdditionalDependencies)
</AdditionalDependencies>
```

Troubleshooting Tips

Include errors → Check `<AdditionalIncludeDirectories>` **Linker errors (LNK2019)** → Check `<AdditionalDependencies>` and `<AdditionalLibraryDirectories>` **Symbol conflicts (LNK2005)** → Use `<IgnoreSpecificDefaultLibraries>` or `/FORCE:MULTIPLE` **Wrong architecture** → Verify `$(Platform)` matches library architecture

Your Specific Changes

1. **Added Content Analysis SDK paths:**

XML

```xml
<AdditionalIncludeDirectories>
  $(CAC_AGENT_INC);$(CAC_GEN_INC);$(PROTOBUF_INC)
</AdditionalIncludeDirectories>
```

2. **Added required libraries:**

XML

```xml
<AdditionalDependencies>
  cac_agent.lib;libprotobuf.lib
```

```
</AdditionalDependencies>
```

3. **Fixed symbol conflicts:**

```xml
<IgnoreSpecificDefaultLibraries>
  legacy_stdio_definitions.lib
</IgnoreSpecificDefaultLibraries>
```

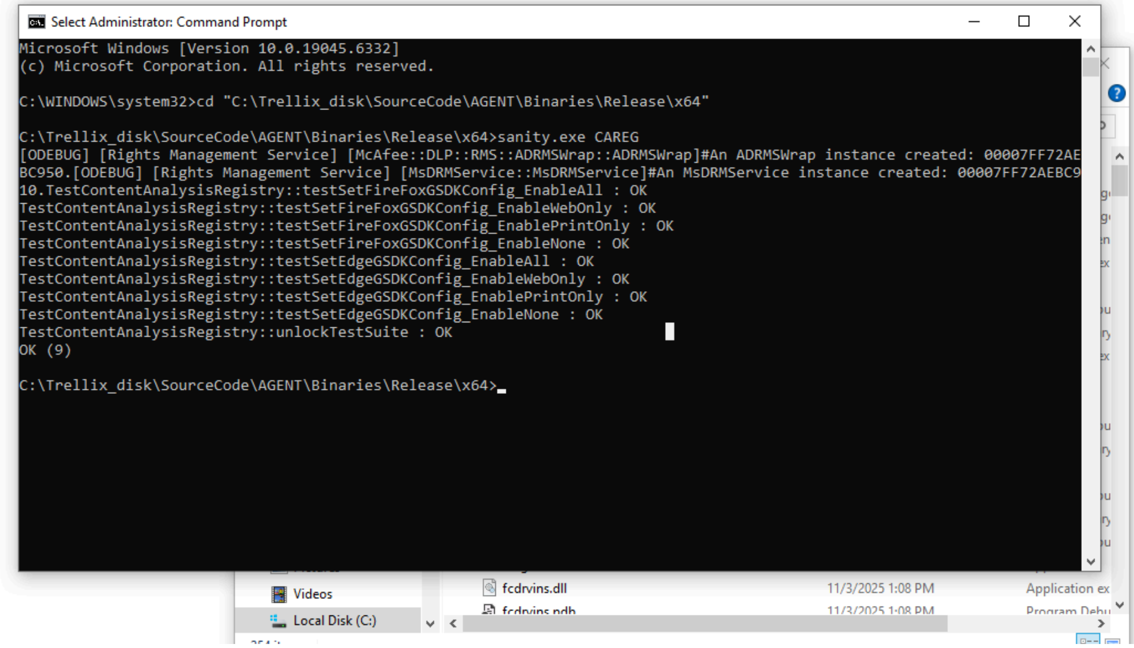
This tells the linker: "Don't link legacy_stdio_definitions.lib because libprotobuf.lib already provides _snprintf symbols."

Here's a summary of the discussion so far:

- Adarsh asked Mallikharjuna about unit test case development for Web protection features, as details were missing in the GR ticket.
- Mallikharjuna explained that Adarsh needs to validate all protection features as part of unit testing. If existing test cases for Web protection rules are available, they can be used; otherwise, new ones need to be built from scratch.
- Mallikharjuna suggested focusing on use cases like file upload, text upload monitoring, and browser URL detection for blacklisted URLs.
- Adarsh confirmed that no present protection test returns exist, and they will need to be created from scratch.
- Mallikharjuna advised Adarsh to document the existing and missing features in Web protection and analyze how to implement unit tests for file and print protection within gsdk.
- Adarsh will focus on file upload, printer, and checking registries for UTs and update the ticket with the criteria.
- Mallikharjuna clarified that setting policies and checking registry settings can be part of the UT workflow.

https://jira.trellix.com/browse/DLPW-18446

Basic UT



Needed to include some libraries, as facing missing external symbols build errors if I don't include.

Existing DLP Unit Test Coverage Areas

Endpoint Protection

- ☑ **File System Protection** (e.g., TestFileSystemProtection.cpp, TestApplicationFileAccessProtection.cpp)
- ☐ **Removable Storage Protection** (e.g., TestRemovableStorageProtection.cpp)
- ☐ **Device Rules/Control** (e.g., TestDeviceRules.cpp, TestDevices.cpp)
- ☐ **Clipboard Protection** (e.g., TestClipboardProtection.cpp)
- ☑ **Screen Capture Protection** (e.g., TestScreenCaptureProtection.cpp)
- ☐ **Printer Protection** (e.g., TestPrinterProtection.cpp)
- ☐ **Cloud Sync** (e.g., TestCloudProtection.cpp, specifically Dropbox)

Network & Web Protection

- ☐ **Network Protection** (e.g., TestNetworkProtection.cpp)
- ☐ **Web Protection (IE)** (e.g., TestIEProtection.cpp)
- ☐ **Email Protection (Outlook)** (e.g., TestOutlookProtection.cpp)

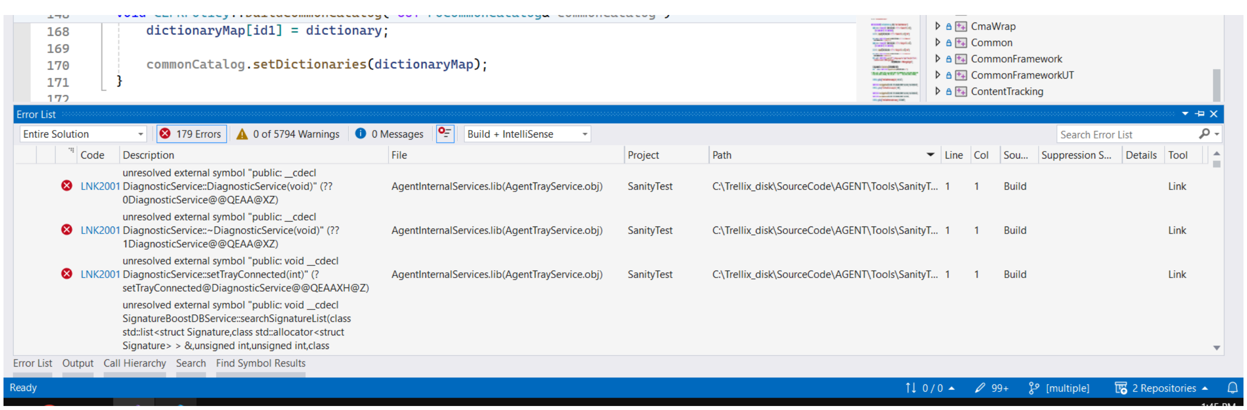
Content Analysis & Classification

- ☐ **Text Extraction** (e.g., TestZTextExtractor.cpp)
- ☑ **Classification Engine** (Keywords, Regex, Dictionaries - e.g., TestClassification.cpp)
- ☐ **Tagging** (Location/Content-based - e.g., TestTag.cpp)
- ☐ **Manual Classification** (e.g., TestManualClassification.cpp)
- ☐ **Hit Highlighting** (e.g., TestHitHighlight.cpp)
- ☐ **Regex Validators** (e.g., TestRegexValidator.cpp)

Agent Core Services & Security

- ☐ **Discovery** (File System, Email - e.g., TestDiscovery.cpp)
- ☐ **Self-Protection** (Agent files/processes - e.g., TestZZZAccessProtection.cpp)
- ☐ **Driver Management** (MiniFilter, Dynamic Install - e.g., TestMiniFilterDriver.cpp)
- ☐ **Security & Encryption** (e.g., TestDLPSecurityService.cpp)
- ☐ **Challenge-Response** (Agent bypass - e.g., TestChallengeResponse.cpp)
- ☑ **Identity/AAD Integration** (e.g., TestEntraIDHelper.cpp)
- ☐ **Configuration** (Advanced Parameters - e.g., TestAdvancedParameters.cpp)

Without adding libraries we face bunch of issues



After adding the library files issues are not seen

https://jira.trellix.com/browse/DLPW-18446

IT

https://jira.trellix.com/browse/DLPW-19969

The existing web protection mechanism, which currently relies on the legacy **DLL-injection method**, is undergoing a critical migration to the modern **GSDK (Generic Software Development Kit) framework**. This new, more robust approach is managed by the `ContentAnalysisConnectorHandler`. The core technical function of this handler involves setting and deleting specific **Windows Registry keys** for target browsers—specifically **Chrome, Edge, and Firefox**—in response to the configured **ePO (ePolicy Orchestrator) policies**. It is important to note that this newly developed code utilizing the GSDK framework is **currently untested**.Unit Testing Strategy

To ensure the reliability of the new GSDK-based web protection, a comprehensive unit testing strategy is being implemented. All new tests will be integrated into the existing sanity test project, utilizing the established **CPP unit test framework**.

Acceptance Criteria for Unit Testing:

The unit tests must be developed to validate the core protection scenarios that are managed directly by the `ContentAnalysisConnectorHandler` (GSDK handler):

1. **File Upload Blocking:** Validation of the mechanism that intercepts and blocks unauthorized file upload operations.
2. **Printer Protection Blocking:** Validation of the mechanism that prevents unauthorized printing operations, a form of data exfiltration protection.

To effectively test these functional use cases in a realistic manner, a supporting utility is required:

- **Browser Action Automation Helper:** A minimal **C++ helper function** must be created.

This function's sole purpose will be to automate standard browser actions (e.g., "launch a browser and upload a file"). This utility will serve as the means to reliably trigger the GSDK analysis and subsequent blocking logic for testing purposes.

-----Integration Testing Strategy (Refinement and Expansion)

In addition to the unit tests, a crucial set of **Integration Tests** must be created to validate the end-to-end functionality, ensuring the GSDK handler interacts correctly with the browser, the policy engine, and the DLP (Data Loss Prevention) logic.

The integration testing will focus on creating dedicated functions for the two primary protection scenarios: File Upload Blocking and Printer Protection Blocking.I. Integration Test: File Upload Blocking

Two separate functions will be created to thoroughly test the file upload blocking scenario for the **Edge and Firefox** browsers.

Scenario and Steps:

1. **Application Launch:** The test must first launch the target browser application (`msh.exe` typically represents the launch command or helper for the test environment).
2. **Website Navigation:** The test must navigate the launched browser to the secure test endpoint: `dlptest.com` and then specifically to the **HTTPS section**.
3. **Policy Preparation (Policy ON Case):**
    - The test must first ensure the necessary policies are active.
    - A **keyword blocking policy** must be pulled from the ePO/Policy Manager, specifically configured to block the keyword "**confidential**."
4. **Test File Preparation:** A simple text file named `confidential.txt` must be created, containing only the single word: "**confidential**."
5. **Test Case 1: Blocking Validation (Policy ON):**
    - The test must attempt to **upload** the `confidential.txt` file to the specified website section.
    - The expected outcome is that the upload **MUST be blocked** by the GSDK protection logic, and the appropriate user/system notification should be verified.
    - **Function 1:** `TestFileUploadBlocking_PolicyOn_Edge()` / `TestFileUploadBlocking_PolicyOn_Firefox()`
6. **Test Case 2: Pass-Through Validation (Policy OFF):**

- The policies must be **un-set** or disabled (i.e., the keyword blocking policy must be removed or deactivated).
- The test must attempt the exact same upload operation of the `confidential.txt` file.
- The expected outcome is that the upload **MUST be successful**, validating that the GSDK connector only enforces protection when a policy is active.
- **Function 2:** `TestFileUploadBlocking_PolicyOff_Edge()` / `TestFileUploadBlocking_PolicyOff_Firefox()`

II. Integration Test: Printer Protection Blocking

Two separate functions will be created to test the printer protection scenario with the exact same two-case criteria (Policy ON vs. Policy OFF) as described for file upload. The focus here is on simulating a printing attempt of the confidential content.

1. **Test Case 3: Blocking Validation (Policy ON):**
   - Ensure the keyword blocking policy for "**confidential**" is **active**.
   - The test must simulate an attempt to **print** a document containing the word "**confidential**."
   - The expected outcome is that the printing operation **MUST be blocked**.
   - **Function 3:** `TestPrinterProtection_PolicyOn_Edge()` / `TestPrinterProtection_PolicyOn_Firefox()`
2. **Test Case 4: Pass-Through Validation (Policy OFF):**
   - Ensure the keyword blocking policy is **not set** or disabled.
   - The test must simulate the same attempt to **print** the document containing the word "**confidential**."
   - The expected outcome is that the printing operation **MUST complete successfully**.
   - **Function 4:** `TestPrinterProtection_PolicyOff_Edge()` / `TestPrinterProtection_PolicyOff_Firefox()`

The immediate next step is to commence development on the four core integration test functions (two for file upload blocking, two for printer protection blocking) to cover both the Policy ON (blocking) and Policy OFF (pass-through) scenarios.

Guidance for Repository Contribution and Code Consistency

When contributing to this repository, it is paramount that all new additions maintain strict consistency with the existing codebase. We emphasize that contributors should **refrain from introducing personal or arbitrary code styles, architectural patterns, or naming conventions** that deviate from the established norms.

Key Principles:

1. **Repository Consistency:** The codebase must present a unified and cohesive structure. This means every new file, module, or function should align seamlessly with the design and conventions already present in the wider repository.
2. **Pattern Adherence:** Before adding new content, contributors are required to thoroughly examine the patterns, structures, and conventions utilized in related and existing files. Your contribution must meticulously follow these same patterns for:
   - **Naming Conventions** (variables, functions, classes, files).
   - **Code Formatting** (indentation, spacing, bracket placement).
   - **Architectural Decisions** (component structure, data flow).
   - **Documentation Style** (comments, docstrings).

Goal:

The primary objective is to ensure that the repository remains easy to navigate, maintain, and scale for all current and future team members. Consistency is a cornerstone of code quality, reducing cognitive load for reviewers and contributors alike, and minimizing the risk of integration errors. **Your compliance with the established patterns is mandatory for pull request acceptance.**

Test Flow for Each Test

Each integration test follows this pattern:

1. **Policy Configuration** - Either enables keyword blocking policy or disables all policies
2. **Registry Validation** - Verifies GSDK registry keys are correctly set/cleared for the browser
3. **Test File Creation** - Creates confidential.txt containing the keyword (for file upload tests)
4. **Manual Test Instructions** - Displays clear step-by-step instructions to the tester
5. **Expected Result Verification** - Clearly states whether blocking or pass-through is expected
6. **Cleanup** - Removes test files and releases resources

Key Features

- **Policy Management**: Uses PolicyBuilder to create keyword-based DLP rules
- **Registry Verification**: Validates GSDK configuration for both Edge and Firefox
- **Test Isolation**: Each test properly sets up and tears down its environment
- **Manual Testing Support**: Since full browser automation is complex, tests provide clear manual

test instructions

- **Extensibility**: Helper methods include TODO comments for future automation implementation

Automation Project: Transitioning from Manual to Semi- or Fully Automatic Process

Objective: The primary goal of this project is the comprehensive implementation of the defined 'to-do' items, focusing specifically on transitioning the current, entirely manual operational process into a more efficient, semi-automatic or fully automatic system.

Scope and Implementation Details:

1. **Core Task Implementation:** Execute all necessary code and logic to complete the remaining 'to-do' list. This includes all prerequisite tasks for automation.
2. **Process Automation Design:**
    - **Feasibility and Strategy:** Investigate and determine the most effective method for establishing either a semi-automatic or a completely automatic workflow. The decision should prioritize long-term stability and efficiency gains.
    - **Technology Constraint (Browser Interaction):** A critical requirement is the need to programmatically interact with a web browser. The implementation for launching and controlling the browser **must** be achieved using C++ code. The developer is tasked with independently researching, selecting, and implementing the most appropriate C++ libraries or system calls to achieve this browser operation.
3. **Code Quality and Efficiency Mandates:**
    - **Minimalism:** The solution must adhere strictly to principles of code minimalism. This means avoiding the creation of superfluous functions, classes, or complex structures. The final code should be as concise and direct as possible while maintaining full functionality.
    - **End-to-End Automation:** The implemented solution must cover the entire process, from the initial trigger to the final necessary output, ensuring a complete, automated workflow without requiring manual intervention at intermediate steps (except where explicitly designed for a semi-automatic process).
4. **Development and Review Protocol:**
    - **Developer Autonomy:** The developer is responsible for all design and implementation decisions regarding the automation strategy and technology choices (especially for the C++ browser interaction).
    - **Deployment and Testing:** Upon completion of the implementation, the developer will submit the final code. The code will then be built, deployed, and subjected to rigorous end-to-end testing by a separate entity to validate the

automation and ensure all project objectives have been met.

Implementation Summary

Automation Approach

Used **Windows SendInput API** for keyboard/mouse automation - minimal, native, no external dependencies.

Key Components Implemented

1. Browser Launch (`launchBrowserAndNavigateToTestSite`)

- Opens Edge/Firefox and navigates to dlptest.com/http-post/
- Checks both Edge installation paths
- 8-second wait for page load

2. File Upload Automation (`automateFileUpload`)

- Uses TAB navigation to reach file input field
- Presses ENTER to open file dialog
- Types full file path using KEYEVENTF_UNICODE
- Submits file selection and upload
- ~5 seconds total execution time

3. Print Automation (`automatePrintOperation`)

- Types "confidential" text using KEYEVENTF_UNICODE
- Sends Ctrl+P to open print dialog
- Auto-submits print with ENTER
- ~4 seconds total execution time

4. Blocking Verification (`verifyBlockingOccurred`)

- Searches for DLP notification windows:
  - "Trellix DLP"
  - "McAfee DLP"
  - "Data Loss Prevention"
  - "DLP Notification"
  - "Content Analysis"
- 5-second polling timeout
- Auto-dismisses notification

5. Success Verification (`verifyOperationSucceeded`)

- Checks no blocking windows appear
- 2.5-second observation period
- Returns PASS if no blocking detected

Test Flow (All 8 Tests)

1. **Setup** → Configure GSDK policy (ON/OFF) + validate registry
2. **Launch** → Open browser to test site
3. **Execute** → Automate file upload OR print operation
4. **Verify** → Check for blocking OR success
5. **Cleanup** → Close browser + delete test files

The project now includes the minimum required set of libraries. The move of the Google content analysis SDK from the agent handler to agent services necessitated the addition of extra libraries and files. This change was implemented to ensure the sanity test project could successfully compile and correctly locate the relevant headers.Although I have only tested the included libraries with the x64 architecture, I have also added these same library types to other architectures to maintain consistency across the project.

Chrome Uninject Issue

Chrome Uninject Issue

Here are the detailed notes from the discussion:

- **Chrome Uninject Issue:** The primary topic of discussion is an issue where the Chrome DLL remains hooked even when gsdk is enabled. Adarsh Anand is currently working on this, facing build errors due to extensive file changes in UT.
- **Previous Fixes and Similar Problems:** Mallikharjuna Yallavula noted a similar problem was previously fixed for a printer handler and that the current issue is related to an HTTP handler.
- **Code Investigation:** The team is examining the logic within `agent handler` and `agent services` (specifically `configuration handler.cpp`) which handles the injection and uninjection of Chrome DLLs.
- **Conflicting Flags and Missing Settings:** A key part of the problem appears to be two potentially conflicting flags: `Chrome hook injected` and `Google Chrome Hook injected`. It was identified that a crucial flag setting was removed, which is believed to be the cause of the current issue.
- **Proposed Solution:** Mallikharjuna Yallavula suggested that Adarsh Anand address the issue by correctly setting these flags and removing any duplicate flags related to Chrome.
- **Further Investigation Required:** Mallikharjuna Yallavula also pointed out that a Boolean pointer intended to reflect or update the correct object might not be working as expected, and further investigation is needed to determine why it's not setting correctly. Adarsh Anand will look into the issue and the provided information.

Based on the video provided, here is a detailed, step-by-step breakdown of all the actions performed.

The video demonstrates a troubleshooting process to see if disabling the "Chrome Web Handler" in a Trellix DLP (Data Loss Prevention) policy successfully unhooks the corresponding DLL (fcagchrome.dll) from the Google Chrome browser.

Step 1: Modify the Trellix DLP Policy

1. The user starts in the **Trellix Policy Catalog** dashboard.
2. They navigate to **DLP Policy 11.13** > **Windows Client Configuration**.
3. The policy "Default Windows Client Configuration (My Default)" is opened for editing.
4. The user scrolls down to the **"Chrome and Edge Web Handler"** section.
5. They **uncheck** the box for **"Chrome Web Handler"**.
6. The user clicks the **"Save"** button (00:05) and waits for the policy to be saved.

Step 2: Launch and Test Google Chrome (First Attempt)

1. The user minimizes the Trellix window.
2. They click the **Google Chrome** icon in the taskbar to launch it.
3. From the "Who's using Chrome?" profile selector, they first click the "Srinath Karugovi" profile (00:15), which opens a window in the background, and then click the **"Work"** profile (00:23), which opens the window they use.
4. In the new Chrome window, they type `dlptest.com/http-post/` into the address bar and press Enter.
5. The page becomes unresponsive. A "Page Unresponsive" dialog appears (00:40).
6. The user clicks the **"Wait"** button.

Step 3: Investigate Chrome's Loaded DLLs

1. While Chrome is unresponsive, the user switches to the **Process Explorer** application.
2. They scroll through the process list and select a `chrome.exe` process.
3. In the bottom pane, which is set to "DLLs" view, they scroll down to check the loaded Dynamic Link Libraries.
4. They find and highlight **`fcagchrome.dll`** (00:56), confirming that the Trellix DLL is still hooked into the Chrome process despite the policy change.

Step 4: Test DLP Functionality in Chrome (Second Attempt)

1. The user switches back to the unresponsive Chrome window and closes it (01:11).
2. They open a new Google Chrome window (this one appears to be a different profile, as the language is German).
3. They type "dlp" in the address bar and select the `dlptest.com/http-post/` URL from the history.
4. The "DLP TEST" website loads successfully this time.
5. The user clicks the **"File Upload"** button.
6. From the "Open" file dialog, they select a file named **`keywords.zip`** and click "Open".
7. Immediately, a **"Trellix Data Loss Prevention"** notification appears (01:25), stating that the web post was blocked. This confirms the DLP is still actively monitoring and blocking uploads from Chrome.

Step 5: Compare with Microsoft Edge DLLs

1. The user switches back to **Process Explorer**.
2. They scroll up the process list and select an **`msedge.exe`** (Microsoft Edge) process.
3. In the DLLs pane, they scroll down and locate **`fcagMSED.dll`** (01:40). This is the equivalent Trellix hook for Microsoft Edge, which was intentionally left enabled in the policy (as seen in Step 1).

Conclusion of the video: The video demonstrates that even after disabling the "Chrome Web

Handler" in the Trellix policy and saving the change, the fcagchrome.dll file is still being loaded by Chrome, and the DLP is still actively blocking web posts.

Sync

Here are the detailed notes from the discussion:

- Mallikharjuna Yallavula is able to reproduce an issue with DLL injection in browsers (Chrome and Edge) on their Windows 11 system, while Adarsh Anand is not experiencing the same.
- The issue seems to stem from browser cache information preventing correct DLL injection or uninjection.
- Mallikharjuna demonstrated that even with Gsdk disabled, the hook remained present, and after enabling Gsdk, the DLL uninjection process reported success, but the DLL was still visible.
- They observed that restarting the browser temporarily resolves the issue, but it can reoccur, suggesting a timing-dependent problem.
- The current Gsdk logic uses

```None
• msedge.exe
```

- as the process name, which differs from the older

```None
• Microsoft Edge gcp.cp.exe
```

- , potentially causing issues with the old browser DLL injector code.
- Mallikharjuna suspects the old logic for injecting/uninjecting DLLs was only for a particular session/user, while the new logic is designed for all sessions, which might be a point of conflict.
- Mallikharjuna believes the problem is with the browser not clearing references, not with their code, as logs indicate successful uninjection.
- Adarsh Anand was unable to locate the log folder under "Program Data/Temp" on their system. Mallikharjuna clarified that the logs are in the "Agent Service" folder within "Program Data/Temp" and asked Adarsh to share the entire logs folder for analysis.
- Adarsh is facing compilation errors after moving logic to a configuration handler, which Mallikharjuna suggested could be resolved by including all necessary paths in the project settings.

Action Items:

- **Adarsh Anand:**
    - Locate the "Agent Service" logs folder under "Program Data/Temp" on the test

    machine.
  - Once the logs are available, update the JIRA ticket with the information that the uninjection call was successful as per the logs.
  - Try including all necessary paths in the project settings to resolve compilation errors.
- **Mallikharjuna Yallavula:**
  - Add more pointers to the JIRA ticket based on their understanding of the browser's role in retaining references.
  - Talk to Ganesh about the issue being a pre-existing one and that they will not be working on it.
  - Update Srinath to share the entire logs folder for analysis.
    Final response to user: Here are the detailed notes from the discussion:
- Mallikharjuna Yallavula is able to reproduce an issue with DLL injection in browsers (Chrome and Edge) on their Windows 11 system, while Adarsh Anand is not experiencing the same.
- The issue seems to stem from browser cache information preventing correct DLL injection or uninjection.
- Mallikharjuna demonstrated that even with Gsdk disabled, the hook remained present, and after enabling Gsdk, the DLL uninjection process reported success, but the DLL was still visible.
- They observed that restarting the browser temporarily resolves the issue, but it can reoccur, suggesting a timing-dependent problem.
- The current Gsdk logic uses `msedge.exe` as the process name, which differs from the older `Microsoft Edge gcp.cp.exe`, potentially causing issues with the old browser DLL injector code.
- Mallikharjuna suspects the old logic for injecting/uninjecting DLLs was only for a particular session/user, while the new logic is designed for all sessions, which might be a point of conflict.
- Mallikharjuna believes the problem is with the browser not clearing references, not with their code, as logs indicate successful uninjection.
- Adarsh Anand was unable to locate the log folder under "Program Data/Temp" on their system. Mallikharjuna clarified that the logs are in the "Agent Service" folder within "Program Data/Temp" and asked Adarsh to share the entire logs folder for analysis.
- Adarsh is facing compilation errors after moving logic to a configuration handler, which Mallikharjuna suggested could be resolved by including all necessary paths in the project settings.

Action Items:

- **Adarsh Anand:**
  - Locate the "Agent Service" logs folder under "Program Data/Temp" on the test machine.
  - Once the logs are available, update the JIRA ticket with the information that the uninjection call was successful as per the logs.
  - Try including all necessary paths in the project settings to resolve compilation errors.
- **Mallikharjuna Yallavula:**
  - Add more pointers to the JIRA ticket based on their understanding of the

browser's role in retaining references.
    - Talk to Ganesh about the issue being a pre-existing one and that they will not be working on it.
    - Update Srinath to share the entire logs folder for analysis.

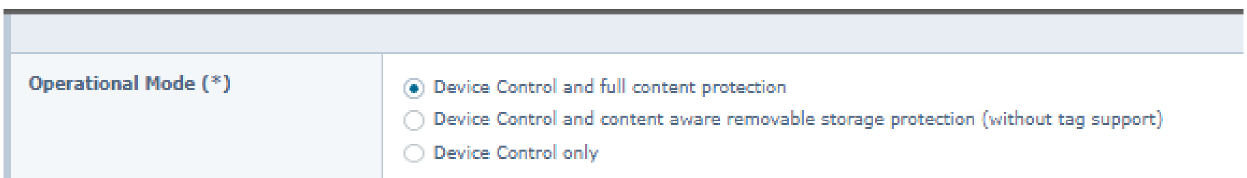
As per the logs observed in the agent service part we can see that an injection of all the chrome hooks get succeeded and the code is working finance expected but somehow the browser is not releasing the dll from its process that's why we are seeing it in process explorer

Debugging Printer

Issue Description: Persistent DLL in Chrome Despite Whitelisting

Here's a summary of the discussion so far:

- Siva Sankari Ganesan outlined a procedure for replacing a DLL, which involves enabling device control, restarting the machine, replacing the DLL, then re-enabling device control and full content protection, followed by another restart.
- Siva Sankari Ganesan explained that after these steps, the process can be tested by attaching to the relevant process (e.g., Chrome or Edge) to verify DLL injection.
- Adarsh Anand questioned the need to attach to a remote process for depacking, to which Siva Sankari Ganesan clarified that attaching to the process is necessary to invoke and check the hook code, which is injected into processes like Chrome or Edge.



- Sign dll orbit > Check for only Device control
- Replace binaries of printer
    - Restart system
- Change config to full content protection
    - Restart system

Dll injected process??
Chrome.exe - remote process attach

Based on my analysis of the 4-minute 22-second screen recording, I have constructed the detailed, step-by-step process document you requested. This audit trail chronicles every significant user interaction to demonstrate the testing and verification of the Trellix Data Loss Prevention (DLP) "Printer Protection" policy.

Detailed Process Document: Trellix DLP Printer Protection Test

This document outlines the step-by-step procedure performed to test the functionality and process-level behavior (specifically DLL hooking) of the Trellix DLP "Printer Protection" feature on Google Chrome and Mozilla Firefox.

Phase 1: Initial State Verification (00:00 - 00:28)

- **00:00 - 00:07**: **Action:** Viewed the active Trellix DLP policy.
  - **Interface:** Trellix Policy Manager.
  - **Navigation:** Policy Catalog > Data Loss Prevention 11.13 > Windows Client Configuration > Default Windows Client Configuration.
  - **Observation:** The setting **"Printer Protection (without tag support)"** under the Device Control section was **checked (enabled)**.
- **00:08 - 00:15**: **Action:** Launched the Google Chrome browser.
  - **Outcome:** A new chrome.exe process was created.
- **00:16 - 00:28**: **Action:** Inspected the running Chrome process.
  - **Interface:** Process Explorer utility.
  - **Details:** Selected the primary chrome.exe process and scrolled through the list of loaded DLLs in the lower pane.
  - **Implied Purpose:** To verify that with the policy enabled, the relevant Trellix DLP printer monitoring DLL was "hooked" (injected) into the chrome.exe process, as suggested by the video's title.

Phase 2: Policy Deactivation and Reactivation Cycle (00:29 - 01:17)

This phase appears to be a rapid toggle to ensure the agent is responsive, culminating in the policy being re-enabled.

- **00:29 - 00:33**: **Action:** Modified the DLP policy.
  - **Interface:** Trellix Policy Manager.
  - **Details: Unchecked** the **"Printer Protection (without tag support)"** setting.
- **00:34 - 00:36**: **Action:** Saved the policy change by clicking the **"Apply"** button.
- **00:41 - 00:48**: **Action:** Forced a policy update on the client.
  - **Interface:** Trellix Agent Monitor utility.
  - **Details:** Clicked the **"Enforce Policies"** button.
  - **Outcome:** The local Trellix agent received the new policy, disabling printer protection.
- **00:49 - 01:00**: **Action:** Briefly reviewed Process Explorer and the Chrome browser.
- **01:01 - 01:09**: **Action:** Re-enabled the DLP policy.
  - **Interface:** Trellix Policy Manager.
  - **Details: Re-checked** the **"Printer Protection (without tag support)"** setting and clicked the **"Apply"** button.
- **01:10 - 01:17**: **Action:** Forced a second policy update on the client.
  - **Interface:** Trellix Agent Monitor utility.
  - **Details:** Clicked the **"Enforce Policies"** button.
  - **Outcome:** The local agent is now enforcing the policy with **printer protection**

re-enabled.

Phase 3: Functional Test of Printer Protection in Chrome (01:18 - 02:24)

This phase tests the real-world function of the enabled policy.

- **01:18 - 01:32**: **Action:** Switched to Process Explorer to observe the `chrome.exe` process DLLs.
- **01:33 - 01:40**: **Action:** Terminated the running Google Chrome instance.
    - **Details:** Right-clicked the Chrome icon in the taskbar and selected **"Close window"**.
- **01:41 - 01:52**: **Action:** Relaunched Google Chrome and immediately switched to Process Explorer.
    - **Implied Purpose:** To confirm that the new `chrome.exe` process, launched *after* the policy was enabled, is correctly hooked by the printer DLL.
- **01:53 - 02:03**: **Action:** Navigated to potentially sensitive content in Chrome.
    - **Details:** Typed the search term **"bombs"** into the address bar and pressed Enter to load the search results.
- **02:04 - 02:16**: **Action:** Attempted to print the webpage.
    - **Details:** Initiated a print job (likely via `Ctrl+P`), which opened the Chrome print preview dialog. The **"Print"** button was clicked.
- **02:17 - 02:24**: **Action: Test Result: Successful Block.**
    - **Outcome:** A **"Trellix Data Loss Prevention"** toast notification appeared in the bottom-right corner.
    - **Notification Message:** "The print was Blocked since sensitive content..."
    - **Conclusion:** This confirms the policy is active and correctly identified and blocked the print job based on content rules.

Phase 4: Policy Deactivation (Round 2) and Firefox Test (02:25 - 03:41)

- **02:25 - 02:45**: **Action:** Disabled the policy again.
    - **Interface:** Trellix Policy Manager.
    - **Details: Unchecked** the **"Printer Protection (without tag support)"** setting and clicked **"Apply"**.
- **02:46 - 02:57**: **Action:** Forced the policy update.
    - **Interface:** Trellix Agent Monitor.
    - **Details:** Clicked **"Enforce Policies"**. The printer protection is now *disabled*.
- **02:58 - 03:07**: **Action:** Launched the Mozilla Firefox browser (labeled "Island").
- **03:08 - 03:18**: **Action:** Inspected the running Firefox process.
    - **Interface:** Process Explorer.
    - **Details:** Selected the `firefox.exe` process and viewed its loaded DLLs.
- **03:19 - 03:30**: **Action:** Switched to the Firefox browser window.
    - **Outcome:** A **"Trellix Data Loss Prevention"** notification appeared.

  - **Notification Message:** "The web post was Blocked since sensitive content..."
  - **Conclusion:** This demonstrates that a *different* DLP rule (Web Post Protection) is active and functioning in Firefox, independent of the printer protection rule that was just disabled.
- **03:31 - 03:41**: **Action:** Closed the Firefox browser.

Phase 5: Final Verification and Policy Restoration (03:42 - 04:22)

- **03:42 - 03:48**: **Action:** Launched Google Chrome again.
- **03:49 - 04:03**: **Action:** Inspected the new Chrome process.
  - **Interface:** Process Explorer.
  - **Details:** Viewed the DLLs for the new chrome.exe process.
  - **Implied Purpose:** To verify that with the printer policy *disabled*, the new Chrome process is *not* hooked by the printer DLL.
- **04:04 - 04:22**: **Action:** Restored the initial policy state.
  - **Interface:** Trellix Policy Manager.
  - **Details: Re-checked** the **"Printer Protection (without tag support)"** setting and clicked **"Apply"**.
  - **Final State:** The session ends with the printer protection policy set to **enabled**.

[SDK] Printer DLL is unexpectedly hooked despite the activation of Printer Protection in Google Chrome and Microsoft Edge SDKs.
Issue Summary:
The printer DLL is being unexpectedly hooked, even with the Printer Protection feature enabled within the Google Chrome and Microsoft Edge SDKs. This suggests a potential conflict or a failure in the intended protection mechanism.
Technical Details:
*   The issue has been observed in the following system configurations:
    *   DLP Ext: Version 11.13.0.17
    *   DLP for Endpoint: Version 11.14.0.45
    *   TA: Version 5.8.3.83 with Latest Msgbus Certs
    *   ePO: 5.10.0 (Build 4098) / SP1 with Update 4 (2.0.0.1739)
    *   Google Chrome: Version 141.0.7390.123
    *   MS Edge: Version 141.0.3537.99
    *   Island: Version 1.72.25 Chromium: 139.0.7258.66
    *   Client: Windows 10E 21H2
Root Cause Analysis:
The root cause has yet to be determined and will be investigated through debugging

tasks.
Next Steps:
* Adarsh Anand has initiated debugging tasks.
* Attached logs and screen recordings for Google Chrome and Edge are available for review.
Request AI assistance for root cause analysis.

Issue Description: Persistent DLL in Chrome Despite Whitelisting

Context: The core issue revolves around the behavior of a specific Dynamic Link Library (DLL) injection mechanism designed to hook printer operations within the Google Chrome browser. A control flag or setting exists to enable this printer hook functionality specifically for Chrome.

Expected Behavior:

1. When the printer hook feature is *enabled* for Chrome, the system is expected to monitor and intercept printer-related calls from the browser.
2. The key mechanism for control is a process whitelist. Once the printer hook is enabled, the Chrome process (e.g., `chrome.exe`) should be automatically added to this internal whitelist.
3. The established protocol dictates that any DLL injected into a process that is subsequently added to the whitelist should be immediately removed or unloaded from that process's memory space. This is typically done to prevent interference or potential conflicts with whitelisted applications.

Observed and Actual Behavior (The Problem):

The expected DLL removal does *not* occur. Even after the enabling flag is checked and Chrome is theoretically added to the whitelist—a condition that should trigger the DLL removal logic—the injected DLL remains persistently loaded within the Chrome process.

Technical Analysis/Root Cause Hypothesis:

The failure lies within the code responsible for processing the whitelist addition and executing the corresponding DLL unload function. Specifically:

- The mechanism to add Chrome to the whitelist appears to be functioning, or at least the condition check is passed.
- However, the subsequent code path intended to enumerate loaded modules in the whitelisted process and forcefully remove the DLL is either failing to execute, encountering a permission/handle error, or the DLL removal logic itself (e.g., using `FreeLibraryAndExitThread` or a similar technique) is flawed or ineffective for the specific DLL/process context.

Summary of Discrepancy:

The system is failing to adhere to the core rule: **Whitelisting a process must result in the removal of the specific monitoring/hooking DLL from that process.** The persistent presence of the DLL contradicts the intended system architecture and may indicate a bug in the whitelist-to-unhooking sequence.

https://github.trellix.com/trellix-products/DLPe.Agent/pull/1472

https://jira.trellix.com/browse/DLPW-19896

Product uninstallation keys removal

Product uninstallation keys removal

When removing the Trellix product in driverinstall.cpp remove the registry entries like these

Chrome config

Windows Registry Editor Version 5.00

; Set up the enrollment token for organizational unit: Trellix
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Google\Chrome]
"CloudManagementEnrollmentToken"="93628663-f462-4487-bdf8-e232fd9e1e33"

Edge config

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Edge]
"OnBulkDataEntryEnterpriseConnector"="[{\"block_until_verdict\": 1, \"default_action\": \"block\", \"enable\":[{\"tags\":[\"dlp\"],\"url_list\":[\"*\"]}], \"minimum_data_size\": 1, \"service_provider\":\"trellix\"}]"
"OnFileAttachedEnterpriseConnector"="[{\"block_until_verdict\": 1, \"default_action\": \"block\", \"enable\":[{\"tags\":[\"dlp\"],\"url_list\":[\"*\"]}], \"service_provider\":\"trellix\"}]"
"OnPrintEnterpriseConnector"="[{\"block_until_verdict\": 1, \"default_action\": \"block\", \"enable\":[{\"tags\":[\"dlp\"],\"url_list\":[\"*\"]}], \"service_provider\":\"trellix\"}]"

Firefox config

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis]
"AgentName"="Trellix DLP Agent"
"AllowUrlRegexList"=""
"DenyUrlRegexList"=""
"PipePathName"="Trellix_DLP"
"AgentTimeout"=dword:0000003c
"BypassForSameTabOperations"=dword:00000001
"DefaultResult"=dword:00000002
"Enabled"=dword:00000001
"IsPerUser"=dword:00000001
"TimeoutResult"=dword:00000000
"ShowBlockedResult"=dword:00000001
"ClientSignature"="Musarubra US LLC"

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Clipboard]

"Enabled"=dword:00000001
"PlainTextOnly"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Downloads]
"Enabled"=dword:00000000

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\DragAndDrop]
"Enabled"=dword:00000001
"PlainTextOnly"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\FileUpload]
"Enabled"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Print]
"Enabled"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\FileUpload]
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Mozilla\Firefox\ContentAnalysis\InterceptionPoints\Print]

Policies.json file

Demo

vfp-azry-xnb (2025-10-27 18:06 GMT+5:30)

Here are the ultra-detailed notes from the discussion:

GSDK and Browser Compatibility:

- **GSDK Capability:** The discussion primarily revolves around the Google Content Analysis SDK (gsdk) and its integration with different browsers.
- **Chrome Enterprise:** GSDK capability is confirmed for Chrome Enterprise.
- **Firefox Island:** Firefox Island will *not* honor the GSDK feature. This needs to be clearly stated.
- **Combinations:** It's crucial to provide examples of combinations. For instance, if only Chrome Enterprise is selected, GSDK will work for Chrome Enterprise, but not for Firefox Island. This clarifies that GSDK is not universally enabled.
- **Existing Web Protection Rule:** The existing web protection rule needs to be detailed in this context.

- **Independent Configurations:** The purpose of independent individual configurations needs to be explained to avoid the misconception that GSDK is enabled for everyone.
- **Versions:** Google Chrome and Island browser versions are currently unknown and will be updated later.

DLP and Image Updates:

- **DLP on Firefox:** When the configuration is enabled, DLP will be visible, and clicking it will show the next DLP on Firefox.
- **Image Addition:** Adarsh needs to add images and corresponding statements to the page to illustrate how DLP appears when the feature is enabled.

Microsoft Edge and Policy Actions:

- **Default Action (Edge):** The default action in Microsoft Edge is "block."
- **Policy Actions:** Existing policies offer "no action" (monitoring), "block request," and "justification" (prompt for justification).

Block Result Pop-up:

- **Show Block Result Policy:** When GSDK is enabled for Chrome, if "show block result" policy is true, a pop-up will appear in the browser (even in Agile).
- **Pop-up Content:** The pop-up will indicate an issue with data (e.g., "data should not be uploaded, protected by Trellix DLP").
- **User Action:** The user needs to click an "OK" button to close the message box.
- **Customer Preference:** Customers often don't want to see this browser pop-up, as DLP already shows a notification.
- **Control over Pop-up:** Microsoft Edge does not have a configuration to control this pop-up. Firefox, however, does have this control and it is configurable.

"Block Until Verdict" and Registry:

- **"Block Until Verdict" (Registry):** Adarsh needs to research Google and Microsoft documentation regarding "block until verdict" (registry value set to 1).
- **Other Values:** Investigate if other values can be set and the reasoning behind setting it to 1.
- **Documentation for QTM:** All this information needs to be updated on the page to prevent issues with the QTM team, ensuring they understand what is currently supported.

Document Consolidation and Older Versions:

- **Duplicate Content:** Adarsh suggested merging two documents due to duplicate content.
- **Mallikharjuna's Reasoning:** Mallikharjuna kept the original document as is because it contains older version-related information (e.g., 11, 12). Removing it might hinder support for customers using those older versions.
- **Current Focus:** The current document is specifically for DLP 11.4.0, Google Content Analysis, SDK integration.

- **Supported Versions Heading:** A "Supported Versions" heading should be added.
- **Island Browser:** Not much information is available for the Island browser, so it can be left to the Island team.

Default Action on DLP Failure:

- **DLP Failure Scenario:** If DLP fails to respond to a data analysis request from the browser, the default action of the browser comes into play.
- **Configurable Default:** If the default action is "block," it will block; if it's "allow," it will allow.
- **DLP Control:** Our DLP does not manage these browser-level default actions, as it only creates registries with fixed values.
- **Future Implementation:** These configurations will be implemented and managed through policies in future releases.
- **Customer Support:** This detailed information is crucial for customer support to answer questions about controlling block messages, as it's not currently controlled by DLP.

Further Exploration:

- **Google/Microsoft Documentation:** Adarsh needs to explore Google and Microsoft documentation for more details on the pop-up configurations.
- **Shivashankari and Srinath:** Adarsh should connect with Shivashankari for Firefox-related updates and with Srinath for his observations on existing releases with Google Content SDK.

Action Items:

- **Adarsh Anand:**
    - Add more details to the gsdk page, including combinations to clarify GSDK capability for Chrome Enterprise and its non-support for Firefox Island.
    - Update the page with images and statements explaining how DLP appears on Firefox when the configuration is enabled.
    - Add bullet points to the notes section explaining GSDK's behavior when enabled/disabled and its interaction with existing web protection rules.
    - Confirm with Shivashankari if the block result pop-up is still showing for Mozilla Firefox and update the documentation accordingly.
    - Research Google and Microsoft documentation regarding "block until verdict" (registry value 1) to understand other possible values and the reasoning behind the current setting.
    - Add a "Supported Versions" heading to the document.
    - Explore more Google and Microsoft documentation regarding pop-up configurations.
    - Connect with Shivashankari for Firefox-related updates.
    - Connect with Srinath to discuss his observations on existing releases with Google Content SDK.
    - Update the Confluence page with all the gathered information.
    - Take ownership of updating Firefox-related information.

- **Mallikharjuna Yallavula:**
    - Connect with Srinath to give him a heads-up about the page updates and provide context.

Here's a summary of the discussion:

The conversation between Mallikharjuna Yallavula and Adarsh Anand focused on the behavior of DLP (Data Loss Prevention) policies, specifically concerning "request for justification" and "no action" configurations, in relation to browser policies (Chrome and Edge).

- **DLP Policy Clarification:** Mallikharjuna clarified that
    - "request for justification" is a DLP policy where the user provides a reason for an action, and this justification is recorded in the event. It does not require admin approval for the action to proceed.
    - Similarly, if a DLP policy is set to "no action," the operation will be allowed even if the browser configuration indicates a block, and the incident will be recorded.
- **Browser vs. DLP Policies:** A key point of discussion was the distinction between browser-level policies (e.g., Chrome's default "Block until verdict") and DLP policies. Mallikharjuna emphasized that DLP policies ultimately control the outcome, even if browser configurations seem to suggest otherwise.
- **Action Item:** Adarsh was asked to create a test case to validate these scenarios in their test machine and update the confluence page based on the results.
- **Future Enhancements:** There was a brief mention of future releases potentially allowing more extensive DLP configurations at the browser level, which could enable the browser to make decisions without needing to consult the DLP system.

Issues

- Mallikharjuna shared a link and discussed an issue where printer hooks are not required when gsdk is enabled for print operations, but Srinath reported the issue is still happening.
- Adarsh will debug the issue in PrinterHandler.cpp and try to reproduce it to find the root cause.
- Mallikharjuna explained that they are rolling back the use of a Trellix-related token for gsdk because customers need to create their own tokens, and Google requires a subscription for these tokens, which customers are unwilling to take.
- As a result, gsdk will not be enabled for Chrome, and customers will be recommended to use the old Web protection rule for Chrome, while Edge, Firefox, and other supporting browsers can use gsdk.
- Adarsh expressed that hands-on experience has helped him understand the code better and that he finds the product and work good, with a large customer base.
- Mallikharjuna highlighted that DLP has a high customer base, including government and security organizations, and that the core technology (C++ running at a system level) offers fast performance compared to other products.
- Mallikharjuna also mentioned some high-profile customers like Boeing, Singapore, Gothic, SBI Bank of Baroda, and Red Bus, noting that Red Bus generates many

escalations.

- Adarsh has one year of experience, having graduated in 2024 and previously worked with Intel.
- Mallikharjuna advised Adarsh to learn as much as possible, as this core technology will enhance his profile.
- Mallikharjuna is currently working on an issue and will connect later.
- They briefly discussed the cyclone and Wi-Fi situation, noting that services were restored but roads still have water.

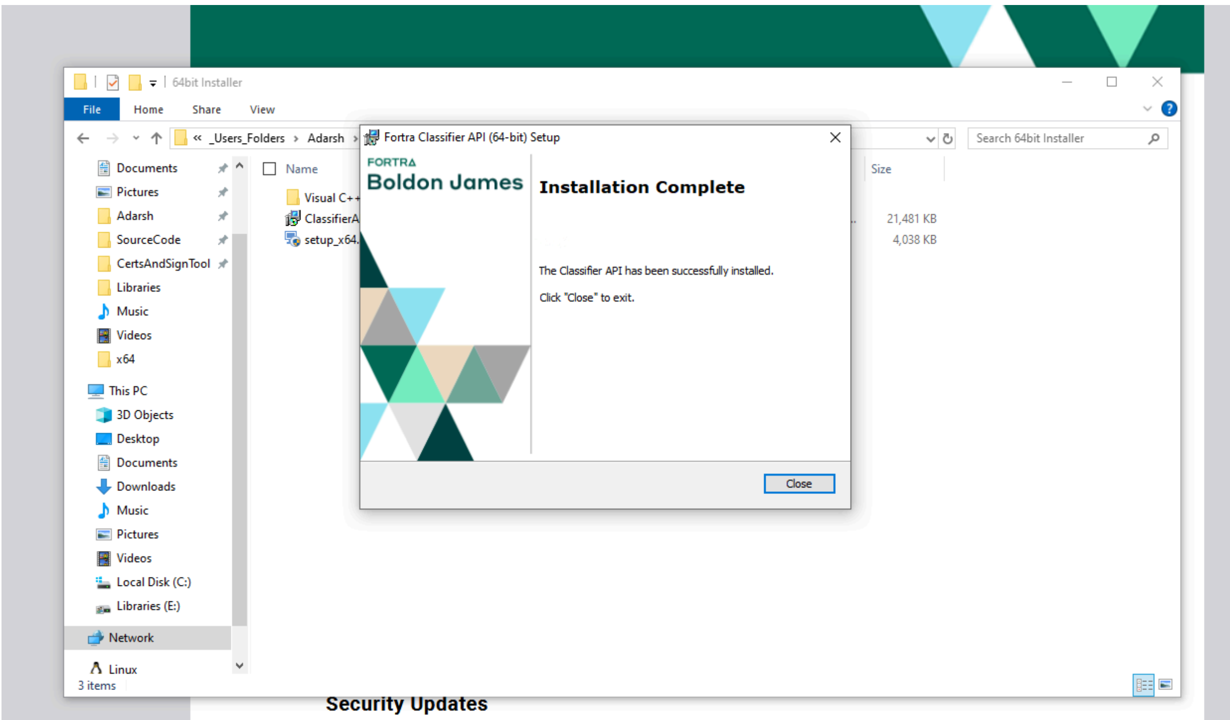
https://jira.trellix.com/browse/DLPW-19790
https://jira.trellix.com/browse/DLPW-19849

Boldon james classification

Bolden James classifications

[DLP Endpoint Training.pptx](#)
[DLP Endpoint & ePO ext.Training .pptx](#)

[Google Drive: Sign-in](#)



Project: SDK Migration and Security Enhancements

- **SDK Migration:**
  - Remove the old SDK and link the new Library API.
  - Update components to use the new SDK (Note: "removed use new SDK!").
- **Security & Compatibility:**
  - Ensure compatibility with Control-Flow Enforcement Technology (CET).
  - Investigate and resolve compatibility issues with CET (specifically, "not CET compatible," "hsp cet," and making "boldon james make it CET compatible").
  - Implement hardware-enforced tag protection.

Migration Summary

Files Modified

1. **AgentBoldonJamesService.h**
   - **Updated interface from `IClassifierAPI*` to `IClassifierAPI1_3*`**
   - **Added `BoldonJamesLabelCachingMode` enum for cache control**
   - **Extended `findBoldonJamesClassifications` with new parameters**
2. **AgentBoldonJamesService.cpp**
   - **Updated ProgID to `BoldonJames.Classifier.API1_3`**
   - **Replaced `GetClassification_2` with `GetClassificationByHandle`**
   - **Fixed all memory leaks (SAFEARRAY and BSTR)**
   - **Added pointer validation and bounds checking**
   - **Handles new return codes (OkNoLabel, UnsupportedFileType)**

The discussion focused on updating the Boldon James library and clarifying repository structures.

- **Boldon James Files for Compilation: The participants agreed that only the `classifier api.tlb` file is required for compilation and should be kept, while other generated files like DLLs and XMLs can be removed from the repository.**
- **DotNet Dependency: Mallikharjuna asked Adarsh to validate whether .NET is required as a prerequisite to run Boldon James and, if so, to identify the minimum supported version for product documentation.**
- **Repository History: Adarsh will create a `Classifier API 1.0` folder to save the old Golden James files from the Libraries folder for historical purposes and add a README file to document its prior use until DLP Windows 114.**
- **Repository Differences: The difference between repositories was clarified: `DLP E dependencies` is for DLP Windows; `DLP dependencies` is a common repository for Windows, Mac, and Discovery; and `DLP third parties` is for source code maintenance (e.g., Open SSL).**
- **Future Package Management: There is a plan to transition to a common, infrastructure-managed package system, similar to npm, to simplify development and version control by eliminating the need for manual library management in the current repositories.**

Boldon James Installation _ Label Creation.mp4

Here are the learning points from the video:

- **Boldon James Installation Components:** The installation of Boldon James requires the Classifier Administrator Server, Classifier API, Email and Office Classifier, and File Classifier.
- **Classifier Administrator Server Installation:** This is a server product that requires enabling IIS (Internet Information Services) on the server OS.
- **Accessing the Administrator Console:** The console can be accessed via `localhost` or the machine's IP address.
- **Label Configuration:** New labels are created under the "Labeling configuration" section in the Administrator Console (e.g., "BJ test").
- **Selector Library:** This library defines the "selector name" and the list of classification "values" (e.g., Confidential, Business, Public). The selector name is crucial and needs to

be noted for later use in the ePO console.

- **Marking Library:** This section is for configuring how the label information is displayed, such as in the header, footer, or as a watermark on documents.
- **Publishing Configuration:** After making changes, the configuration must be published. This can be done via Active Directory or a manual file store.
- **Exporting Configuration Files:** Upon publishing, the latest configuration files are updated in the `C:\ProgramData\Boldon James\Config Publish` folder, which needs to be copied to the client machines.
- **Client Machine Prerequisites:** For Boldon James to work on the client machine and integrate with DLP, three software components must be installed: File Classifier, Email and Office Classifier, and the **Classifier API**.
- **Classifier API Importance:** The Classifier API is essential for integration with Trellix DLP. If it is missing, the DLP agent logs will show a "metadata API error."
- **Configuring the Client Machine Registry:** To make the client machine use the exported configurations, specific string and DWORD values must be created under the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Boldon James\Config Manager` (and its 6432 node counterpart). These keys specify the label configuration name (`BJ test`), the policy name (`Commercial Policy`), and the local path (`C:\BJ config`) where the configuration files are stored.
- **Applying Labels on the Client:** Once configured, users can right-click a file and use the File Classifier option to select a classification value, which updates the file's label and changes its overlay icon.
- **DLP Integration (Brief Mention):** The classified files can then be governed by DLP policies based on the assigned Boldon James labels, though the configuration of the DLP policy itself is covered in a separate video.

🎬 Boldon James Label Configuration in Trellix DLP.mp4

Here are the learning points from the video:

- **Configuring Boldon James Labels in Trellix DLP:** The video demonstrates the process of configuring Boldon James labels to be detected by Trellix DLP (Data Loss Prevention) policies.
- **Boldon James Administration Console:** It shows the Boldon James Classifier Administration Console, where labels and policies are initially created under the "labeling configurations" section.
- **Selector Name and Values:** A key concept is the "Selector Library," which contains a "selector name" (e.g., "testjutu") and its corresponding "values" (e.g., "public," "business," "confidential," and "internal").
- **Creating Classification in EPO:** To integrate with Trellix DLP, a new classification must be created in the McAfee ePolicy Orchestrator (ePO) console, navigating to the "classification section."
- **Using Third-Party Tags:** The classification uses the "third party tags" option, which is designed to support Boldon James and Titus tags.
- **Mapping Selector Name to Tag Name:** The "selector name" from the Boldon James console is entered as the "Boldon James tag name" in the ePO classification

configuration.

- **Specifying the Label Value:** The specific label value (e.g., "confidential") that DLP should detect, monitor, or block is entered in the classification.
- **Creating a DLP Policy Rule:** The newly created classification is then added to a DLP policy rule (e.g., a "web protection rule") with a defined reaction (e.g., "block").
- **Enforcing the Policy:** Pending changes in ePO must be applied to enforce the updated policy to the endpoint machines.
- **Prerequisites for Endpoint:** For the third-party tag detection to work on the endpoint, the following components are required:
    - Boldon James File Classifier (and typically Email/Office Classifier).
    - Boldon James Classifier API (which is critical and might sometimes be missing).
- **Testing the Configuration:** The video shows a demonstration where a file labeled as "business" is successfully uploaded (not blocked), but when the same file is re-labeled as "confidential" (the value configured in the policy), the DLP policy blocks the upload to VirusTotal.
- **Troubleshooting:** If DLP is not working, the first steps are to ensure the configuration is proper: the selector name is added correctly, and the file has the correct label value. The second check is to confirm the required prerequisites, especially the Classifier API, are installed on the endpoint.

https://github.trellix.com/trellix-products/DLPe.Agent/pull/1504

The current objective is to thoroughly review all designated Bolden James classification files and subsequently develop a comprehensive set of unit test cases for all implemented functions. This involves extracting relevant data from all specified files. Two dedicated files, `test_bolden_james.h` and `test_bolden_james.cpp`, have been created within the `tools/sanity_test` directory for this purpose. Utilizing the file naming conventions as a guide, all necessary unit tests must be written employing the CPP unit testing framework. Upon completion, the solution will be built, and the correctness of the tests will be verified. Please commence work by analyzing the codebase to gain an understanding of the existing code structure before proceeding to write all required tests using the CPP unit framework.

https://confluence.trellix.com/spaces/DE/pages/104829158/Boldon+James+Setup+for+DLP+Endpoint [old]

\\10.140.165.198\qafiler\software\Boldon James

Enable-WindowsOptionalFeature -Online -All -FeatureName IIS-NetFxExtensibility45, IIS-ASPNET45, NetFx4Extended-ASPNET45, IIS-DefaultDocument, IIS-HttpCompressionDynamic, IIS-HttpErrors, IIS-ManagementConsole, IIS-LoggingLibraries, IIS-WindowsAuthentication

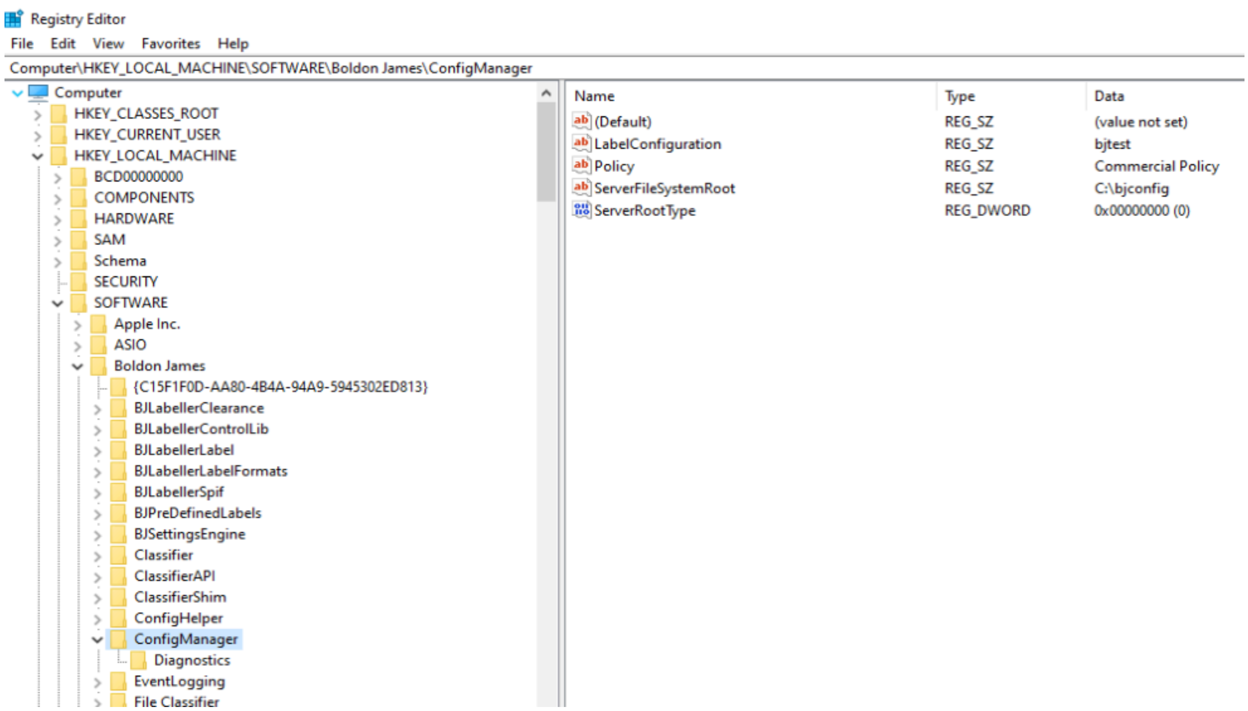
Configuring Registry Settings

The application requires specific configurations to be present in the Windows Registry. These must be applied to both the standard 64-bit hive and the 32-bit compatibility hive (WOW6432Node).

2a. Standard 64-bit Registry Configuration

The following script creates the `ConfigManager` key and necessary values under the standard software path.

PowerShell



None

```
# Define the Registry Path
$regPath = "HKLM:\SOFTWARE\Boldon James\ConfigManager"

# Create Key if missing
if (!(Test-Path $regPath)) {
    New-Item -Path $regPath -Force | Out-Null
    Write-Host "Created Path: $regPath" -ForegroundColor Green
```

```
}


# Add Values

New-ItemProperty -Path $regPath -Name "LabelConfiguration" -Value "bjtest" -PropertyType String -Force | Out-Null

New-ItemProperty -Path $regPath -Name "Policy" -Value "Commercial Policy" -PropertyType String -Force | Out-Null

New-ItemProperty -Path $regPath -Name "ServerFileSystemRoot" -Value "C:\bjconfig" -PropertyType String -Force | Out-Null

New-ItemProperty -Path $regPath -Name "ServerRootType" -Value 0 -PropertyType DWORD -Force | Out-Null


Write-Host "Registry update complete." -ForegroundColor Cyan
```

2b. 32-bit (WOW6432Node) Registry Configuration

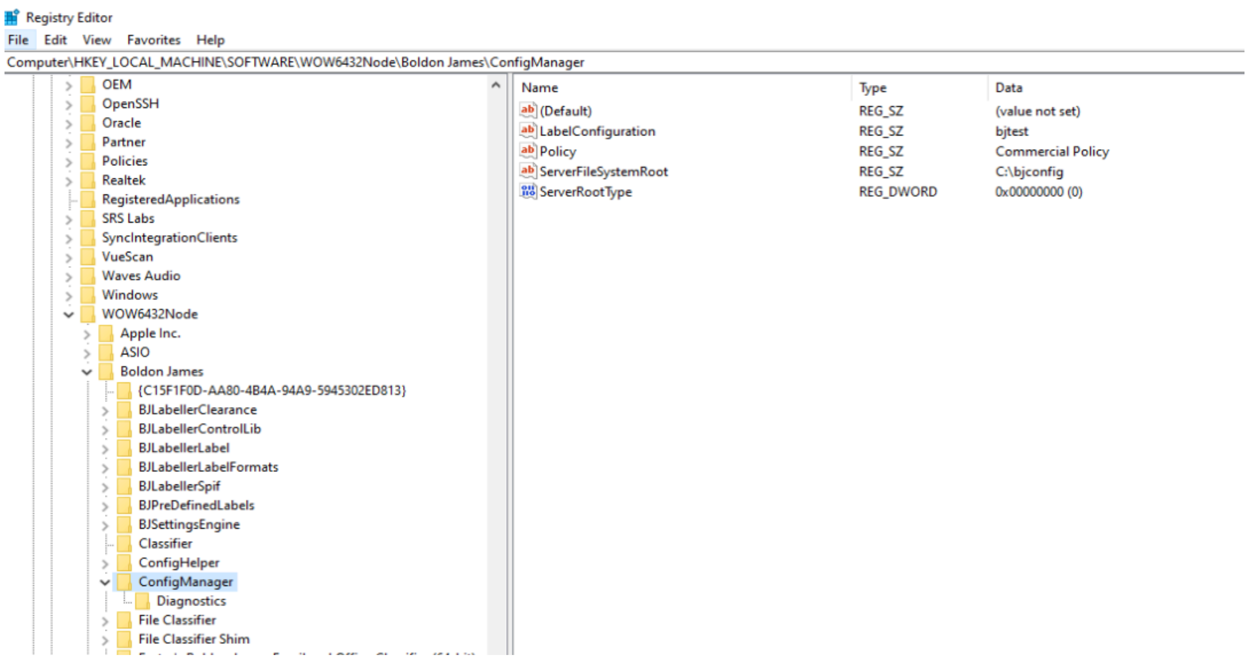
This script ensures that 32-bit applications running on the 64-bit OS can access the same configuration data.

PowerShell

None

```
# Define the 32-bit Node Registry Path

$regPath = "HKLM:\SOFTWARE\WOW6432Node\Boldon James\ConfigManager"


# Create Key tree if missing

if (!(Test-Path $regPath)) {

    New-Item -Path $regPath -Force | Out-Null

    Write-Host "Created Path: $regPath" -ForegroundColor Green

}
```

```
# Add Values

New-ItemProperty -Path $regPath -Name "LabelConfiguration" -Value
"bjtest" -PropertyType String -Force | Out-Null

New-ItemProperty -Path $regPath -Name "Policy" -Value "Commercial
Policy" -PropertyType String -Force | Out-Null

New-ItemProperty -Path $regPath -Name "ServerFileSystemRoot" -Value
"C:\bjconfig" -PropertyType String -Force | Out-Null

New-ItemProperty -Path $regPath -Name "ServerRootType" -Value 0
-PropertyType DWORD -Force | Out-Null


Write-Host "WOW6432Node Registry update complete." -ForegroundColor
Cyan
```



Feature Name	Function

IIS-NetFxExtensibility45	Enables IIS to use .NET code to extend its functionality.
IIS-ASPNET45	Installs the ASP.NET engine for running modern .NET web apps.
NetFx4Extended-ASPNET45	Activates the required underlying .NET 4.x framework libraries.
IIS-DefaultDocument	Allows the server to automatically load a default file (e.g., index.html).
IIS-HttpCompressionDynamic	Compresses data on the fly to speed up website loading.
IIS-HttpErrors	Enables the display of user-friendly, custom error pages.
IIS-ManagementConsole	Installs the graphical user interface for managing IIS.
IIS-LoggingLibraries	Installs tools needed to record server access logs.
IIS-WindowsAuthentication	Allows users to log in using integrated Windows credentials.

All installation steps are complete, and file classification using "public," "business confidential," and "internal" tags is successful.

Able to do the full workflow of Bolden James without upgrading the SDK code.