White Paper

Cisco AMP for Endpoints:

Exploit Prevention

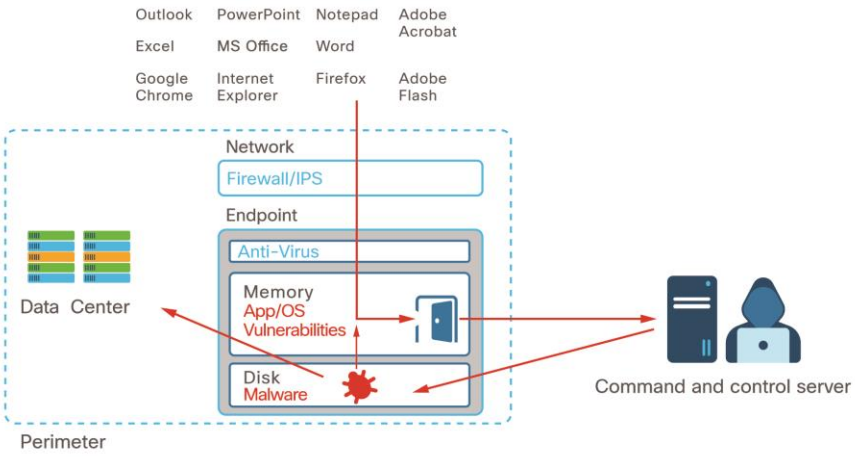
Contents

What you will learn

This document explains the preventative security engine added to Cisco® Advanced Malware Protection (AMP) for Endpoints as a part of AMP Connector version 6.0.5 (and enhanced with version 6.2.1) for Windows—Exploit Prevention. The document is intended to provide a technical explanation of the technology as well as help assess the value of Exploit Prevention as an augmentation of the security stack available with the product.

Introduction

Memory attacks penetrate via endpoints and malware evades security defenses by exploiting vulnerabilities in applications and operating system processes. A majority of these attacks operate in the memory space of the exploited application and remain untouched by most security solutions once they gain access to the memory.



AMP for Endpoints Exploit Prevention provides an integral preventative security layer for protecting endpoints, servers, and virtual environments from file-less and memory injection attacks, as well as obfuscated malware. It does so by changing the static nature of the defense landscape to a dynamic one and makes it more difficult for attackers to plan and execute successful attacks.

AMP for Endpoints protection lattice

Cisco AMP for Endpoints protection capabilities comprise several technologies that work together to prevent, detect, and remediate malicious code at the endpoint.

The core in-memory prevention technologies include:

- **Exploit Prevention** defends endpoints from memory attacks commonly used by obfuscated malware and exploits targeting software vulnerabilities of protected processes.
- **System Process Protection** defends critical Windows system processes from being tampered with or compromised through memory injection attacks by other offending processes.

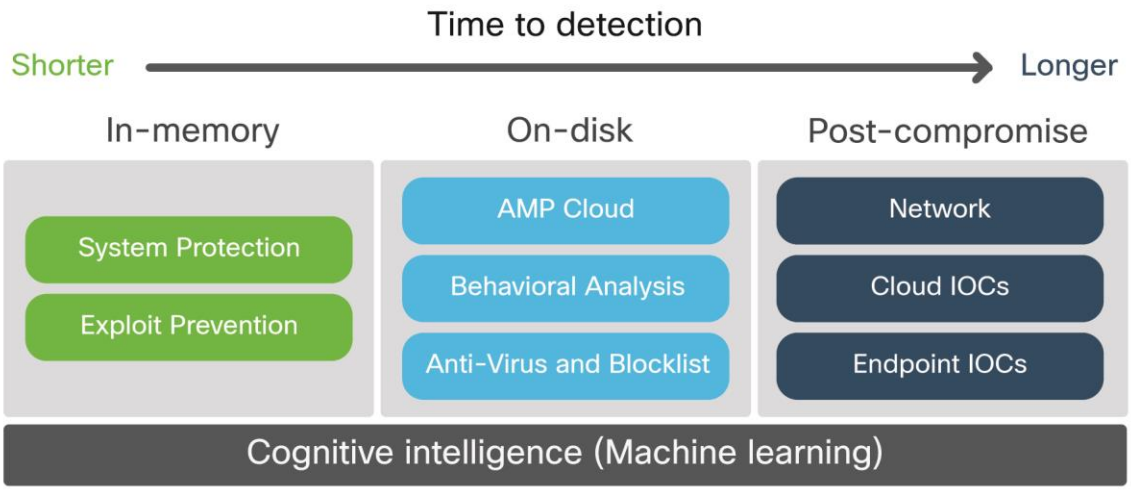
The core on-disk detection technologies include:

- **AMP Cloud** blocks malware using the global threat intelligence that is constantly augmented with new threat knowledge from Cisco Talos™, Cisco Threat Grid, and Cognitive Intelligence research.
- **TETRA** is a traditional signature-based antivirus engine, resides on the endpoint, and provides on-disk malware detection capabilities; TETRA is a part of the AMP Connector for Windows.
- **Malicious Activity Protection (MAP)** offers runtime detection and blocking of abnormal behavior associated with a running file or process (for example, ransomware-related behaviors).
- **Custom Detections** serve the goal of delivering robust control capabilities to the security analyst by providing the ability to define custom signatures and enforce blacklists using industry-standard formats.

The core post-infection detection technologies include:

- **Device Flow Correlation** inspects incoming and outgoing network communications of a process/file on the endpoint and allows the enforcement of a restrictive action according to the policy.
- **Cloud Indicators of Compromise** help surface suspicious activity observed on the endpoints through pattern recognition; related alerts serve as the trigger for more in-depth investigations and response.
- **Endpoint IOCs** are a powerful threat-hunting capability for scanning post-compromise indicators across multiple endpoints and can be imported from custom open IOC-based files.

Cognitive Intelligence further enhances AMP for Endpoints efficacy through the efforts of Cisco's machine learning research team (12+ years of research experience, 70+ Machine Learning data scientists and engineers, 60+ patents and fillings, and over 200 publications). With that, the product is enriched with file-less and agent-less detection capability based on the analysis of network (web logs, NetFlow) telemetry. The result of that analysis is context-rich threat knowledge tailored to a particular organization. Cognitive Intelligence also allows for correlating threat behaviors and activity across multiple monitored datasets (network, endpoint, attacker model), thereby providing an increased level of security efficacy (cross-layer analysis). On top of that, other dedicated models operate embedded inside the AMP and Threat Grid products to deliver ML-based static file analysis capability.



These security capabilities are the foundation of the overall approach to pervasive advanced malware protection. While Cisco recommends using all of these engines in conjunction with each other to leverage the full value of the product, customers can select whether to enable or disable one or another feature through a policy. Exploit Prevention technology, which is the focus of this white paper, is itself just one of the important elements of functionality that Cisco AMP for Endpoints delivers. While listed separately, these technologies work together as a protection lattice to provide improved visibility and increased control across the entire attack continuum.

Additional functionality of AMP for Endpoints, such as static and dynamic file analysis with Threat Grid, and retrospective security is well described in the user guide located at https://docs.amp.cisco.com.

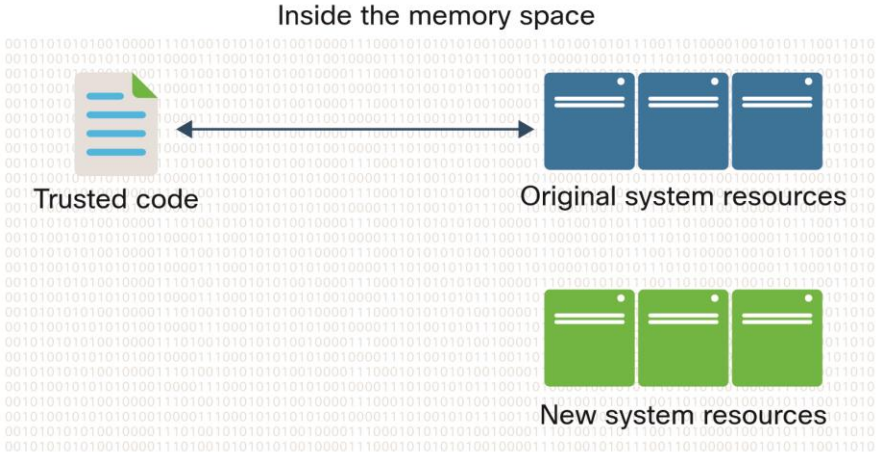
Exploit Prevention technology

Exploit Prevention protects against malware and exploits that target unpatched or zero-day vulnerabilities to run malicious code. It employs a profound understanding of the way in which a Windows process operates, and follows its various mechanisms. In the current release (AMP Connector version 6.2.1 and later for Windows), Exploit Prevention protects a set of preconfigured 32-bit and 64-bit applications listed below. The AMP for Endpoints research and engineering team constantly looks for ways to enhance anticipated protection levels and offer support for an extended list of processes. Exploit Prevention is a preventative security engine that provides broad protection against a variety of threat types. It does so through memory manipulations of a protected process as described below.

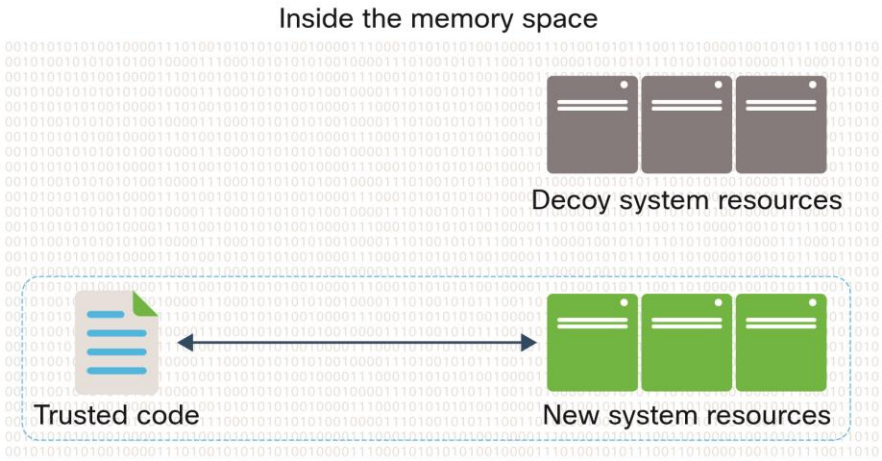
How it works

Let's break down how Exploit Prevention works into three simple steps.

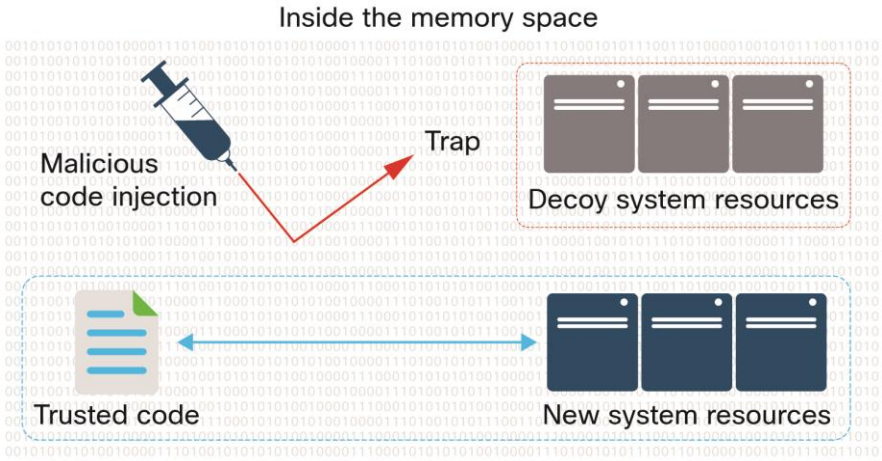
Step 1: Each time a user opens a protected application, Windows loader loads into memory all resources required to run this application. Exploit Prevention appends a tiny DLL (Dynamic-link library) to the Windows loader, which proactively changes the process structure. In other words, locations of libraries, variables, functions, and other data elements are modified in a coordinated manner. Exploit Prevention scrambles all the locations of resources in the memory. In most cases, the scrambling is done in one-way randomization without a key. That means there is no way to reverse engineer the locations of resources in the new scrambled memory, which makes the memory unpredictable for attackers.



Step 2: After creating the new memory structure, the next step is to make the legitimate application code aware of the new locations of its required resources. The application continues to run normally and works without any change to its behavior. At the same time, Exploit Prevention creates a decoy of the original memory structure that can be used as a trap for malicious code. With that, the tiny DLL appended by Exploit Prevention stops working, practically exhibiting no activity.



Step 3: When malicious code targets the original memory structure, it's unaware of the changes made by Exploit Prevention. The code will seek the original, predictable target it was designed to exploit, but instead will hit a decoy, and be neutralized as a result. Access to read-only memory for execution or writing is easily detected, and the application doing so is malware by definition, and as such, it is immediately stopped and trapped. It is prevented from executing and terminated as early as possible in the kill chain: before command and control communication and before the payload has been dropped on the disk.



To summarize, Exploit Prevention offers a true preventative security engine that doesn't require policy tuning or prior knowledge of threats or written rules to operate. Once a process is under protection, the protection will continue until the process is terminated. When Exploit Prevention stops the attack, it stops the application from running and logs contextual data along with event classification in AMP for Endpoints Device Trajectory and Events view (which can also be queried via AMP APIs). It's also worth mentioning that there is no concept of audit mode with Exploit Prevention, unlike with some of the other AMP for Endpoints engines.

Exploit Prevention is a part of AMP Connector for Windows. Refer to the release notes for details on the supported operating systems.

Protected processes

Starting with AMP for Windows Connector version 6.2.1, Exploit Prevention protects the following 32-bit and 64-bit processes. Child processes of the protected applications inherit protection.

- Microsoft Excel application
- Microsoft Word application
- Microsoft PowerPoint application
- Microsoft Outlook application
- Internet Explorer browser
- Mozilla Firefox browser
- Google Chrome browser
- Microsoft Skype application
- TeamViewer application
- VLC media player application
- Microsoft Windows Script Host
- Microsoft Powershell application
- Adobe Acrobat Reader application
- Microsoft Register Server
- Microsoft Task Scheduler Engine
- Microsoft Equation Editor

The protection plan also includes the following critical system processes. In this case, the protector is injected into the running processes.

- Local Security Authority Subsystem
- Windows Explorer
- Spooler Subsystem

Support for other processes may be added in the future releases of AMP Connector for Windows. Please refer to the release notes and the user guide for the latest details.

Performance

Performance is a large part of the endpoint security selection criteria. AMP for Endpoints adds little overhead to the system performance. Exploit Prevention engine doesn't explicitly imply any performance penalty or any changes to the end user experience. Exploit Prevention doesn't do hooks that perform various comparisons on every operation to detect attacks. After the engine has completed the memory scrambling, the number of executions and the number of jumps to reach a particular DLL or execute a particular function (for example) is preserved at the same count.

The only impact that can be considered is memory impact caused by the decoy. However, because the authorized application code does not use those decoy resources during normal activity, they are flashed back to the kernel and therefore do not have any performance impact. Those resources are not in the user workset; consequently, it is not a real penalty for virtual memory (unless there is malware being blocked because of accessing the decoy; in such situation, a penalty is generally acceptable). Another possible impact is application load time because of the scrambling performed. As a result, a protected application may load a little (5% to 10%) slower than usual, but it doesn't imply any runtime penalty or impact to the user experience. Exploit Prevention does not have any effect on the CPU performance.

Compatibility

Compatibility with software installed on the endpoint is an essential aspect of any endpoint security solution. Enabling Exploit Prevention engine has potential compatibility issues only with some anti-exploit products that perform memory manipulations on the same processes. While Cisco does not recommend running more than one memory protection solution at the same time, some workarounds can help co-exist. Excluding incompatible processes from the list of applications that other anti-exploit products protect is the key to successful deployment and testing.

Microsoft Enhanced Mitigation Experience Toolkit (EMET) is included in Windows 10 and higher by default and is available as a downloadable add-on for Windows 7. To improve compatibility between the Exploit Prevention component of Cisco AMP for Endpoints and Microsoft EMET, consider the following:

- For Windows 7 endpoints, disable EAF (Export Address Table Access Filtering) rule for processes protected by AMP for Endpoints Exploit Prevention engine.
- For Windows 10 endpoints, disable EAF (Export Address Table Access Filtering), IAF (Import Address Table Access Filtering), and ACG (Arbitrary Code Guard) rules for processes protected by AMP for Endpoints Exploit Prevention engine.

Please refer to the AMP for Endpoints user guide (https://console.amp.cisco.com/docs) for compatibility details and further guidance on the subject matter. Contact the Cisco technical support team for assistance, if necessary.

Compare Exploit Prevention with ASLR

Address Space Layout Randomization (ASLR) is a computer security technique, which involves randomly positioning the base address of an executable and the position of libraries, heap, and stack in the address space of a process. The random mixing of memory addresses performed by ASLR means that an attack no longer knows at what address the required code (such as functions of Returned-Oriented Programming [ROP] gadgets) is actually located. That way, rather than removing vulnerabilities from the system, ASLR attempts to make it more challenging to exploit existing vulnerabilities.

ASLR has certain limitations that Exploit Prevention helps overcome:

- Boot time–based randomization: The base addresses of DLLs are randomized at boot time only; that may be exploited by combining vulnerabilities such as memory disclosure or brute force attacks.
- Unsupported executables/libraries: An executable must be built with ASLR support; otherwise, there is no protection. This limitation is mostly observed in older versions of Windows (prior to Windows 8).
- ASLR does not trap the attack: ASLR aims to prevent an attack from reliably reaching its target memory space only. However, once the shellcode jumps to the wrong address during exploit (due to the memory randomization), the program behavior is undefined (possible exception is a crash).
- ASLR does not alert in the case of an attack: When a vulnerability is exploited and fails due to ASLR's memory randomization, no alert or attack indication is received.
- ASLR does not provide information about an attack: Evidence that is often crucial for a forensic investigation (such as exploited processes, memory dumps, call stacks) is not provided with ASLR.
- A number of attack techniques have been developed by attackers to bypass ASLR, including the use of ROP chains in non-ASLR modules, JIT/NOP (Just-In-Time or NO-OP) spraying, memory disclosure vulnerabilities, and other techniques.

The Exploit Prevention approach is different from ASLR. While the concepts may sound similar, ASLR lacks several important elements to make it successful at countering zero-day exploits and targeted attacks.

Compare Exploit Prevention with EMET

Microsoft EMET (Enhanced Mitigation Experience Toolkit) is a toolkit for Windows that aims to address the same problem space; however, EMET only looks for known attacks using configurable rules.

EMET	Exploit Prevention
EMET used for exploit detection has several flaws: • Explicit rules are defined to detect specific types of attacks (rule per set of attacks), which means attackers can bypass EMET if they understand these rules • Applications must be configured to work with EMET explicitly • Has compatibility issues with several applications as it blocks behaviors that those applications require. Therefore many of the rules in EMET should be disabled, decreasing protection • Requires a large amount of RAM; system reboots are required to apply changes, which impacts performance • No forensic data on blocked attacks* • Can be bypassed in Windows 7, 8, 10** • Offers exploitation-only protection	Exploit Prevention takes a completely different approach that is prevention focused: • Fully proactive prevention that is not rule based; there is no way to reverse engineer the locations of resources in the new scrambled memory • No prior knowledge of the attack is required: any access to nonmorphed area is considered malicious • Compatible with most security solutions and doesn't have application compatibility issues • No runtime components and no runtime performance penalty, only load time • Provides detailed forensic information through the AMP for Endpoints console; can be extracted through the APIs • Part of an enterprise-grade security solution • Protects against exploitation, post-exploitation, and malware

*Lack of forensic information: EMET is not designed to deliver visibility into security events.

**EMET can be bypassed by various methods. For example:

- Executing 64-bit shellcode inside 32-bit processes
- Disarming EMET through unhooked methods
- Utilizing EMET .dll to bypass ASLR and hooking

Threats, malware, and exploit techniques

As an integral part of AMP for Endpoints, Exploit Prevention implements an in-depth defense mechanism that is aimed at preventing various attack vectors at initial stages of the attack chain. As a result, the engine stops the following threats, malware, and exploit techniques (when used to exploit processes on the protection list), which are often observed as components of sophisticated threat campaigns. The list is not exhaustive and represents examples, which are presented in three categories:

- Exploitation
- Post-exploitation
- Malware

EXPLOITATION		
Memory corruption exploits	Many exploits fall under the category of memory corruption. Memory corruption refers to **stack/heap overflows, integer overflows, use after free, type confusion,** and many others. Buffer overflows occur when a program attempts to put more data in a buffer than it can hold, or when an application attempts to write data in a memory area outside of the boundaries of a buffer. Integer overflow condition occurs when an integer value is incremented to a value that is too large to store in the associated representation. This becomes security critical, when the result is used to control looping, determine memory allocation, or make another security decision. Use after free assumes that the memory in question is allocated to another pointer validly at some point after it has been freed and the original pointer to the freed memory is used again and points to somewhere within the new allocation. Such techniques often allow a remote attacker to execute arbitrary code or crash a vulnerable application. Memory corruption is part of an exploitation process. Using the memory corruption, the attacker would like to achieve read-and-write priviledge (read and write to memory). Eventually, the attacker would want to hijack the flow of the vulnerable application and redirect the application code to injected malicious code.	Memory corruption exploits require an attacker to know the memory layout of the vulnerable process to be able to exploit it successfully. Exploit Prevention scrambles the address locations of resources in memory, their names, order, etc. Due to this scrambling, even though the exploit may have read primitive, it fails to find the real indicators and therefore fails to hijack the flow. Instead, it locates the deceptively injected resource's stubs (links, names, sections, etc.).
ROP/return to lib	As part of the exploitation process, the exploit needs to bypass mechanisms like ASLR and DEP (Data Execution Prevention). These protection techniques have been embedded inside the Windows OS starting from XP SP3. Within the past couple of years, most of the popular applications have been almost fully adapted to utilize these mechanisms as a part of their full source recompilation, although some are yet to recompile their Windows 10 code to allow the enforcement of the mechanism on all processes. Attackers adapted and succeeded in bypassing this mechanism. One of the techniques that is often used as part of an exploit is ROP. Instead of executing the malicious code from nonexecutable memory, the malicious code will be composed of legitimate executable code chunks that are allowed to execute. The process of looking up and executing those chunks/gadgets is called ROP and is achieved by manipulation of the stack.	With Exploit Prevention, the legitimate chunks of code are in an unpredictable location. Instead, Exploit Prevention injects traps in the predictable locations and therefore prevents the execution of the ROP.
Heap spraying	Heap spraying is utilized as a part of two different stages of the exploitation process: (1) Some of the exploit types utilize weaknesses in memory management of the process. Many of the use-after-free exploits will be able to trigger just after the memory manager starts to enable memory optimizations due to the fragmentation that is caused following a specific type of heap spraying technique. (2) As part of a flow redirection process, the exploit needs to locate the next stage-injected code. In order to make the location of this code predictable, the exploit sprays the heap with the malicious code chunks so that the redirection to the hard-coded address can succeed.	Exploit Prevention makes the memory management less predictable due to the allocation of additional deceptive memory regions in different locations of the heap.

POST-EXPLOITATION			
Shellcode		Shellcode (or position-independent code) is designed to be copied into memory at an arbitrary memory address and executed from that address. The key part for shellcode is finding critical functions within memory (for example, load a module from disk, open socket for communication, get data from Internet, read from a file, etc.). A common tactic for shellcode is to locate the address of the kernel32.dll module; then it locates the kernel32!LoadLibrary function address to identify less granular functions and the kernel32!GetProcess function to reflectively load a module within the memory without touching the disk.	Exploit Prevention tricks shellcode into using a deceptively injected kernel32.dll and other core skeleton modules, while the real modules are randomized, changed, and no longer called the same name, located at the same address, or expose similar function names, etc.

POST-EXPLOITATION			
Code injection	Remote thread injection	This is the basic and the most popular technique that is currently used to inject malicious code into a whitelisted process. The technique involves a process of: • Allocating executable memory within the target process VirtualAllocEx • Writing into the executable memory the malicious code • Executing CreateRemoteThread, which creates a new thread in the target process (this thread will start its execution from the entry to the malicious code) The malicious code will either load a follow-up module from disk with additional functionality or will stay fileless and may decrypt additional malicious code into the memory. the malicious code needs access to the process API to execute its intention.	Exploit Prevention helps mitigate it during the first steps of the malicious code execution. The malicious code will fail to locate the randomized API and will be trapped during the execution of the deceptively injected API.
Code injection	Atom bombing	Atom bombing is a technique that uses atom tables for writing into the memory of another process and forces a legitimate program to retrieve the malicious code from the table. Legitimate processes containing the malicious code can be manipulated to execute that code. Atom bombing is not associated with an application or operating system vulnerabilities or bugs, so there's nothing to exploit or patch. This technique relies on how OS mechanisms are designed (and affects all current Windows OS flavors, including Windows 10). As part of the flow, the malicious code that is injected needs to execute ROP to copy itself into executable memory in order to bypass DEP. ROP needs to use executable gadgets within the memory that are in the exact location or offset within its own memory.	Exploit Prevention helps mitigate this technique for protected processes in a number of ways (as part of its in-depth defense approach), both preventing the ROP execution and following the execution of the API as part of the post-exploitation flow. Example detection: Dridex Banking Trojan (Talos Threat Round-up for the Week of Mar 20-24)
Code injection	Minimalist code injection	Minimalist code injection is not an entirely new class of code injection techniques. It's more of an evolution of existing process injection techniques. It offers the following improvements over known techniques: • Requires only four Windows API calls to achieve code injection • Does not require the created process to be created in a suspended state • Does not require new threads to be created in the target process This makes it more difficult for security solutions to detect it.	Exploit Prevention mitigates and prevents the execution of the injected code and is therefore indifferent to the technique used to inject the code (the same way that it prevents most of the other code injection techniques). Talos blog: IcedID Banking Trojans Teams up with Ursnif/Dreambot for Distribution
Process hollowing	Basic	Process hollowing is a technique in which malicious code unmaps the legitimate code from memory of the target process (NtUnmapViewOfSection) and overwrites the memory space of the target process (like explorer.exe, svchost.exe, or others) with a malicious executable (VirtualAllocEx). The result of process hollowing is a process that looks legitimate on the outside but is primarily malicious on the inside. The newly introduced malicious code needs to identify the memory layout of the wrapper hollow process in order to properly function. Only then it can execute.	Exploit Prevention randomizes the wrapper process memory structure before parts of its inner memory are replaced with a malicious code. This causes the execution failure of the injected code.
Process hollowing	Process doppelganging	Process doppelganging is a novel process injection technique that is similar in effect to process hollowing, although it uses a different set of Windows API calls. To achieve process injection using doppelganging, a malicious process creates a transacted file using a benign executable. The transacted file is overwritten with malicious code. A section is created using the transacted file, and then the transaction is rolled back. Then a low-level, outdated API call is used to create a process using the section, and the process environment is manually populated. The main thread of the process is resumed, causing the malicious code to execute and completing the process of code injection. Due to its file-less nature and the use of low-level API calls, this technique is often successful in evading endpoint security technologies.	Exploit Prevention randomizes the wrapper process memory structure before parts of its inner memory are replaced with a malicious code. This causes the execution failure of the injected code.
Reflective loading		Reflective DLL injection refers to loading a DLL into a host process from memory rather than from disk without using the Windows loader. The library is responsible for loading itself by implementing a minimal custom-written portable executable file loader. It can then govern how it will load and interact with the host. This technique allows loading a library to the target process without registering with the process itself. In many cases, the malicious code needs to have access to mapping functions like GetProcAddress. These critical functions are used to map the imported functions of the injected DLL into the exported functions of the host target process.	Exploit Prevention allows the mitigation of reflective loading by randomizing the API of the host process. The malicious code will execute deceptive stubs instead of the real mapping functions.

MALWARE		
Packer-based malicious attacks	Packer-based malware is malware whose code is obfuscated and compressed using various techniques, making it hard to detect using traditional antivirus signatures. Packers are used by attackers to obfuscate the code's real intention, avoid traditional antivirus signatures, and protect the code from being reverse engineered. However, a majority of attacks use multiple stages, and at some point, unpacking should happen to download and execute the next malicious payload.	Due to the need to bypass runtime detection solutions and hooking, at the point of unpacking and deobfuscation, the malware tries to locate and execute system calls directly from the memory without using imported API. This API is randomized, so, instead, the malware executes the deceptively injected API. This helps the engine block and trap the attack before malicious payloads are dropped on the disk.
Adware	Adware is unwanted software that automatically renders advertisements in order to generate revenue for its author. Advertisements may be in the user interface of the software, on a screen presented to the user during the installation process, or in a browser. The functions may be designed to analyze visited Internet sites, present advertising content, install additional programs, and so on. Adware can be bundled with a software or a game that the user wants. In many cases, the bundle will access a third-party server during installation to deliver the most current adware or add-on without it touching the disk. This adware is reflectively loaded directly into the process memory. Additionally, the same adware installers are prone to hijacking of the delivery mechanism and can be utilized to deliver a much higher-severity code.	Depending on how the software is written, Exploit Prevention can identify and block only the component that is performing the adware injection and not the whole software bundle. This is dependent on if the adware injector dll is a separate component from the rest of the software (usually licensed by adware distributor). In that case, Exploit Prevention just blocks the adware DLL from loading (hence effectively performing adware cleansing).

Note: The techniques described above do not represent an exhaustive list of exploit techniques that are defeated by Exploit Prevention. Exploits that make assumptions about the memory layout of the process and that utilize vulnerabilities associated with remote code execution are prevented. This helps the address such attacks as target exploit attacks, exploit kit attacks, ransomware attacks, Trojan attacks, in-memory backdoor attacks (for example, utilizing macros written in PowerShell or VBScript), and many others.

Summary

Exploit Prevention introduces a different security stack, much earlier in the attack phase, which complements the protection lattice of AMP for Endpoints. The benefit is that it occurs early in the kill chain, reducing reliance on front-end knowledge of the attack and rule tuning, as well as patching of vulnerabilities and cleanup. Cisco still recommends establishing a process for timely vulnerability assessment and patching to ensure stronger security posture. Exploit Prevention's method of remapping memory modules within the address space of protected processes and altering key memory structures allows Exploit Prevention to successfully prevent a variety of modern and sophistication exploitation, post-exploitation, and malware techniques.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **https://www.cisco.com/go/offices**.

Printed in USA C11-742008-00 03/19