BinSkim: A Windows PE (Portable Executable) Security Correctness Tool.

Contents

[Description of Tool 1](#_Toc469292104)

[Source and Drop Location 1](#_Toc469292105)

[Running BinSkim from the Command-Line 1](#_Toc469292106)

[Quick Start 1](#_Toc469292107)

[Command-Line Argument Reference 2](#_Toc469292108)

[Help command 2](#_Toc469292109)

[Analyze Command 2](#_Toc469292110)

# Description of Tool

BinSkim is a checker that examines portable executables and their associated PDBs to identify various security problems. These include:

* **Use of outdated compiler toolsets.** Binaries should be compiled against the most recent compiler toolsets wherever possible to maximize use of current compiler-level and OS-provided security mitigations.
* **Insecure compilation settings.** Binaries should be compiled with the most secure settings possible, to enable OS-provided security mitigations, maximize compiler error and warnings reporting, etc.
* **Signing issues**. Signed binaries should be signed with cryptographically strong algorithms.

# Source and Drop Location

BinSkim is a Github-hosted [open-source project](https://github.com/Microsoft/binskim/). The latest version of the tool is always published as a [NuGet package](https://www.nuget.org/packages/Microsoft.CodeAnalysis.BinSkim/).

# Running BinSkim from the Command-Line

## Quick Start

The primary function of BinSkim is to analyze Windows portable executables (.dlls, .exes, etc). To analyze a file, pass one or more arguments that resolve one or more portable executables.

// Analyze a single binary named MyProjectFile.dll found in c:\temp  
// and emit verbose messages during analysis  
binskime.exe analyze c:\temp\MyProjectFile.dll –verbose

// Analyze all files with the .dll or .exe extension starting in the  
// current working directory and recursing through all child directories  
binskim analyze \*.exe \*.dll –recurse

// Analyze all files with the .dll extension starting in the current  
// current directory and write results to a SARIF log file  
binskim analyze \*.dll --output MyLog.sarif

## Command-Line Argument Reference

### Help command

The following command-lines invoke the general BinSkime help message. This message will display all the built-in ModernCop commands (help, analyze, capture, et al) for which more detailed help can be requested:

binskim.exe --help

To request detailed help for specific commands, invoke ‘binskim.exe help [command]’, eg:

binskim.exe help analyze  
binskim.exe help exportRules  
binskim.exe help exportConfig  
binskim.exe help dump  
binskim.exe help version

### Analyze Command

The ‘analyze’ command supports the following additional arguments:

|  |  |
| --- | --- |
| Argument (short form, long form) | Meaning |
| --sympath | Symbols path value, e.g., SRV\*http://msdl.microsoft.com/download/symbols or Cache\*d:\symbols;Srv\*http://symweb |
| -o, --output | File path to which SARIF-formatted analysis output will be written. |
| -v, --verbose | Emit verbose output. The resulting comprehensive report is designed to provide appropriate evidence for compliance scenarios. |
| -r, --recurse | Recurse into subdirectories when evaluating file specifier arguments. |
| -c, --config | (Default: ‘default’) Path to policy file that will be used to configure analysis. Passing value of 'default' (or omitting the argument) invokes built-in settings. |
| -q, --quiet | Do not log results to the console. |
| -s, --statistics | Generate timing and other statistics for analysis session. |
| -h, --hashes | Output SHA-256 hash of analysis targets when emitting SARIF reports. |
| -e, --environment | Log machine environment details of run to output file. WARNING: This option records potentially sensitive information (such as all environment variable values) to any emitted log. |
| -p, --plug-in | Path to plug-in that will be invoked against all targets in the analysis set. |
| --help | Displays this table of argument information. |
| --version | Displays BinSkim version details. |

In addition to the named arguments above, BinSkim accepts one or more specifiers to a file, directory, or filter pattern that resolves to one or more binaries to analyze. arguments can include wildcards, relative paths (in which case the file or directory path is resolved relative to the current working directory), and environment variables. All these arguments can be applied one or more times on the command-line. For analysis to occur, at least one specifier must be passed that resolves to one or more files.

#### --sympath

The --sympath argument provides a path to a symbol server. The syntax for this argument is identical to the symbol path provided to Windows debuggers, as documented at <https://msdn.microsoft.com/en-us/library/windows/hardware/ff558829(v=vs.85).aspx>. As per this documentation, the symbol path can also be used to specify a directory location to cache any downloaded symbols.

NOTE: BinSkim requires PDBs to complete a significant subset of its analysis (see list below) which generally should be located alongside a target .dll or .exe. BinSkim explicitly clears any symbol path configured in the environment via %\_NT\_SYMBOL\_PATH% to prevent unexpected network activity or slowdowns related to PDB acquisition during analysis.

When BinSkim cannot properly load a PDB, because it is missing, corrupted, etc., the tool will emit an instance of ERR97. This message will report the problem including information on the specific HRESULT (and its meaning) error code returned by the PDB loading API, e.g.:

error ERR997.ExceptionLoadingPdb : BA2013 : 'symsrv.dll' was not evaluated for check 'InitializeStackProtection' because its PDB could not be loaded. ‘(E\_PDB\_NOT\_FOUND (File not found))’

The following table lists all BinSkim rules by Id and name and details the specific PDB information examined during analysis. Generally, each of these checks also inspects each object module language in order to restrict analysis to Microsoft C/C++ compilers.

|  |  |  |
| --- | --- | --- |
| **Id** | **Name** | **Data Examined** |
| BA2006 | BuildWithSecureTools | Compiler version of all linked object modules |
| BA2014 | DoNotDisableStackProtectionForFunctions | IDiaSymbol::get\_isSafeBuffers value for all binary functions |
| BA2002 | DoNotIncorporateVulnerableDependencies | Source files for all linked object modules |
| BA2007 | EnableCriticalCompilerWarnings | Compiler warning level and explicitly disabled warnings for all linked object modules |
| BA2011 | EnableStackProtection | IDiaSymbol::get\_hasSecurityChecks for all linked object modules |
| BA2013 | InitializeStackProtection | Scans PDB for /GS feature function name |

#### -o, --output

The **-o or --output argument** specifies a file path to which BinSkim’s [SARIF](https://rawgit.com/sarif-standard/sarif-spec/master/Static%20Analysis%20Results%20Interchange%20Format%20(SARIF).html)-formatted results will be written. The [Microsoft SARIF SDK](https://github.com/microsoft/sarif-sdk) ships with a Microsoft Visual Studio Add-In that can be compiled and used to load SARIF log files into the Microsoft VS IDE.

#### -v, --verbose

By default, BinSkim output is restricted to errors and warnings. BinSkim can also be configured to provide more comprehensive output by passing **-v** or --**verbose** on the command-line. In this case, BinSkim will emit explicit messages for each rule as it examines each target, including whether a binary passed the check successfully or if the check was skipped because a target was not applicable to analysis.

#### -r, --recurse

The -r or --recurse argument will recurse into child directories for each file specifier passed on the command-line. If the argument does not appear on the command-line, each file specifier will be resolved against the provided directory, if there is one, or the current working directory, if there is not.

#### -c, --config

The -c or --config argument can be used to pass settings, rendered as XML, that can be used to reconfigure analysis.