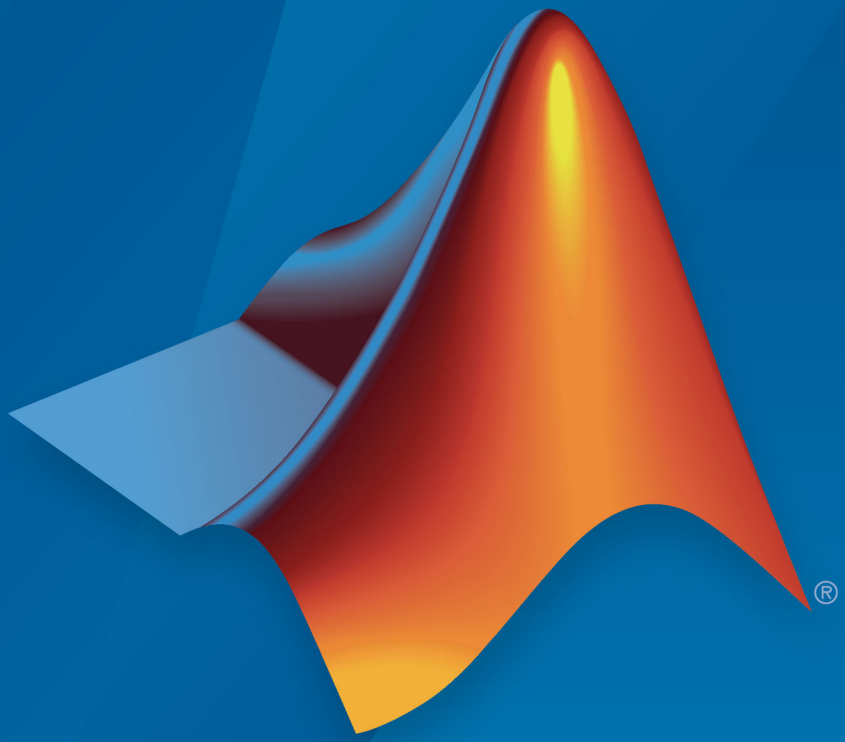
Polyspace® Bug Finder™ Server™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Polyspace® Bug Finder™ Server™ Release Notes

Contents

R2019b

R2019b

Version: 3.1

New Features

Bug Fixes

Compiler Support: Set up Polyspace analysis easily for code compiled with Cosmic compilers

Summary: If you build your source code by using Cosmic compilers, in R2019b, you can specify the compiler name for your Polyspace® analysis.

You specify a compiler using the option `Compiler (-compiler)`.

```
polyspace-bug-finder-server -compiler cosmic -target s12z -sources file.c ....
```

Benefits: You can now set up a Polyspace project without knowing the internal workings of Cosmic compilers. If your code compiles with your compiler, it will compile with Polyspace in most cases without requiring additional setup. Previously, you had to explicitly define macros that were implicitly defined by the compiler and remove unknown language extensions from your preprocessed code.

AUTOSAR C++14 Support: Check for misuse of lambda expressions, potential problems with enumerations, and other issues

In R2019b, you can look for violations of these AUTOSAR C++14 rules in addition to previously supported rules.

AUTOSAR C++14 Rule	Description	Polyspace Checker
A0-1-4	There shall be no unused named parameters in non-virtual functions.	`AUTOSAR C++14 Rule A0-1-4`
A3-1-2	Header files, that are defined locally in the project, shall have a file name extension of one of: `.h`, `.hpp` or `.hxx`.	`AUTOSAR C++14 Rule A3-1-2`
A5-1-2	Variables shall not be implicitly captured in a lambda expression.	`AUTOSAR C++14 Rule A5-1-2`
A5-1-3	Parameter list (possibly empty) shall be included in every lambda expression.	`AUTOSAR C++14 Rule A5-1-3`

AUTOSAR C++14 Rule	Description	Polyspace Checker
A5-1-4	A lambda expression shall not outlive any of its reference-captured objects.	`AUTOSAR C++14 Rule A5-1-4`
A5-1-7	A lambda shall not be an operand to `decltype` or `typeid`.	`AUTOSAR C++14 Rule A5-1-7`
A5-16-1	The ternary conditional operator shall not be used as a sub-expression.	`AUTOSAR C++14 Rule A5-16-1`
A7-2-2	Enumeration underlying base type shall be explicitly defined.	`AUTOSAR C++14 Rule A7-2-2`
A7-2-3	Enumerations shall be declared as scoped enum classes.	`AUTOSAR C++14 Rule A7-2-3`
A16-0-1	The preprocessor shall only be used for unconditional and conditional file inclusion and include guards, and using the following directives: (1) `#ifndef`, (2) `#ifdef`, (3) `#if`, (4) `#if defined`, (5) `#elif`, (6) `#else`, (7) `#define`, (8) `#endif`, (9) `#include`	`AUTOSAR C++14 Rule A16-0-1`
A16-7-1	The `#pragma` directive shall not be used.	`AUTOSAR C++ 14 Rule A16-7-1`
A18-1-1	C-style arrays shall not be used.	`AUTOSAR C++ 14 Rule A18-1-1`
A18-1-2	The `std::vector<bool>` specialization shall not be used.	`AUTOSAR C++ 14 Rule A18-1-2`
A18-5-1	Functions `malloc`, `calloc`, `realloc` and `free` shall not be used.	`AUTOSAR C++ 14 Rule A18-5-1`

AUTOSAR C++14 Rule	Description	Polyspace Checker
A18-9-1	The `std::bind` shall not be used.	`AUTOSAR C++ 14 Rule A18-9-1`

For all supported AUTOSAR C++14 rules, see “AUTOSAR C++14 Rules” (Polyspace Bug Finder Access).

CERT C++ Support: Check for pointer escape via lambda expressions, exceptions caught by value, use of bytewise operations for copying objects, and other issues

In R2019b, you can look for violations of these CERT C++ rules in addition to previously supported rules.

CERT C++ Rule	Description	Polyspace Checker
DCL59-CPP	Do not define an unnamed namespace in a header file	`CERT C++: DCL59-CPP`
EXP61-CPP	A lambda object shall not outlive any of its reference captured objects.	`CERT C++: EXP61-CPP`
MEM57-CPP	Avoid using default operator new for over-aligned types	`CERT C++: MEM57-CPP`
ERR61-CPP	Catch exceptions by lvalue reference	`CERT C++: ERR61-CPP`
OOP57-CPP	Prefer special member functions and overloaded operators	`CERT C++: OOP57-CPP`

For all supported CERT C++ rules, see “CERT C++ Rules” (Polyspace Bug Finder Access).

CERT C Support: Check for undefined behavior from successive joining or detaching of the same thread

In R2019b, you can look for violations of these CERT C rules in addition to previously supported rules.

CERT C Rule	Description	Polyspace Checker
CON39-C	Do not join or detach a thread that was previously joined or detached	`CERT C: Rule CON39-C`

For all supported CERT C guidelines, see “CERT C Rules and Recommendations” (Polyspace Bug Finder Access).

New and updated Bug Finder defect checkers

Summary: In R2019b, you can check for new issues and also see improved results for previous checkers.

New Checkers in R2019b

Defect	Description
`Unnamed namespace in header file`	Header file contains unnamed namespace leading to multiple definitions
`Lambda used as decltype or typeid operand`	`decltype` or `typeid` is used on lambda expression
`Operator new not overloaded for possibly overaligned class`	Allocated storage might be smaller than object alignment requirement
`Bytewise operations on nontrivial class object`	Value representations may be improperly initialized or compared
`Missing hash algorithm`	Context in EVP routine is initialized without a hash algorithm
`Missing salt for hashing operation`	Hashed data is vulnerable to rainbow table attack
`Missing X.509 certificate`	Server or client cannot be authenticated
`Missing certification authority list`	Certificate for authentication cannot be trusted
`Missing or double initialization of thread attribute`	Noninitialized thread attribute used in functions that expect initialized attributes or duplicated initialization of thread attributes

Defect	Description
`Use of undefined thread ID`	Thread ID from failed thread creation used in subsequent thread functions
`Join or detach of a joined or detached thread`	Thread that was previously joined or detached is joined or detached again

Updated Checkers in R2019b

Defect	Description	Update
`Pointer or reference to stack variable leaving scope`	Pointer to local variable leaves the variable scope	The checker now detects pointer escape via lambda expressions.

MISRA C:2012 Directive 4.12: Dynamic memory allocation shall not be used

Summary: In R2019b, you can look for violations of MISRA C®:2012 Directive 4.12. The directive states that dynamic memory allocation and deallocation packages provided by the Standard Library or third-party packages shall not be used. The use of these packages can lead to undefined behavior.

See `MISRA C:2012 Dir 4.12`.

Configuration from Build System: Compiler version automatically detected from build system

Summary: In R2019b, if you create a Polyspace analysis configuration from your build system by using the `polyspace-configure` command, the analysis uses the correct compiler version for the option `Compiler (-compiler)` for GNU® C, Clang, and Microsoft® Visual C++® compilers. You do not have to change the compiler version before starting the Polyspace analysis.

Benefits: Previously, if you traced your build system to create a Polyspace analysis configuration, the latest supported compiler version was used in the configuration. If your code was compiled with an earlier version, you might encounter compilation errors and might have to specify an earlier compiler version before starting the analysis.

For instance, if the Polyspace analysis configuration uses the version GCC 4.9 and some of the standard headers in your GCC version include the file `x86intrin.h`, you can see a compilation error such as this error:

```
/usr/lib/gcc/x86_64-linux-gnu/6/include/avx512bwintrin.h, line 2427:
                                    error: invalid type conversion
|    return (__m512i) __builtin_ia32_packssdw512_mask ((__v16si) __A,
|
```

You had to connect the error to the incorrect compiler version, and then explicitly set a different version. Now, the compiler version is automatically detected when you create a project from your build command.

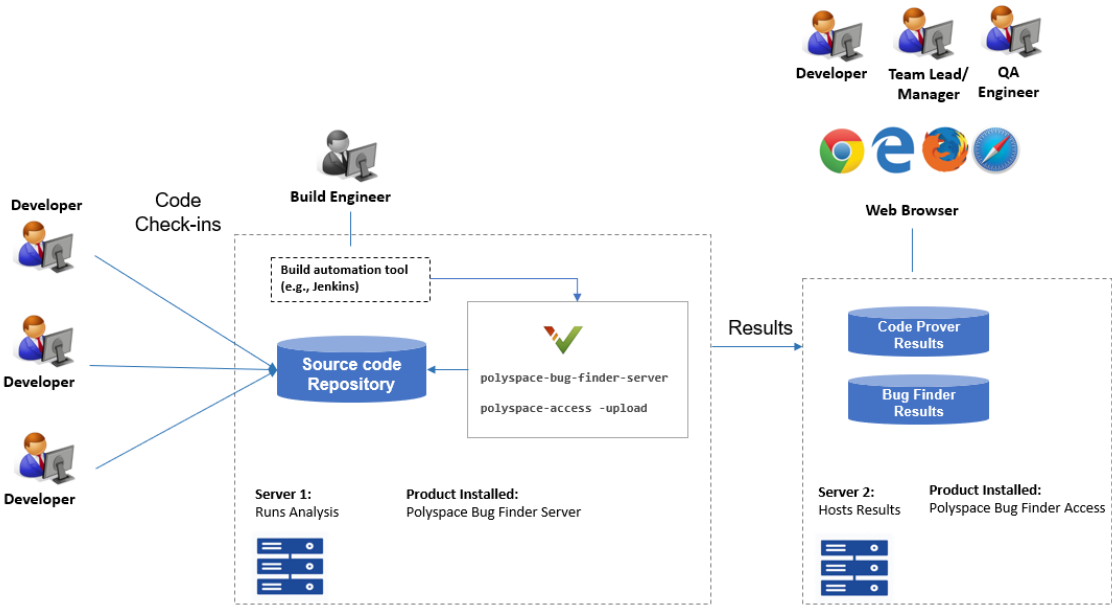
R2019a

Version: 3.0

New Features

Bug Finder Analysis Engine Separated from Viewer: Run Bug Finder analysis on server and view the results from multiple client machines

Summary: In R2019a, you can run Bug Finder on a server with the new product, Polyspace Bug Finder™ Server™. You can then host the analysis results on the same server or a second server with the product, Polyspace Bug Finder Access™. Developers whose code was analyzed (and other reviewers such as quality engineers and development managers) can fetch these results from the server to their desktops and view the results in a web browser, provided they have a Polyspace Bug Finder Access license.



Note: Depending on the specifications, the same computer can serve as both Server 1 and Server 2.

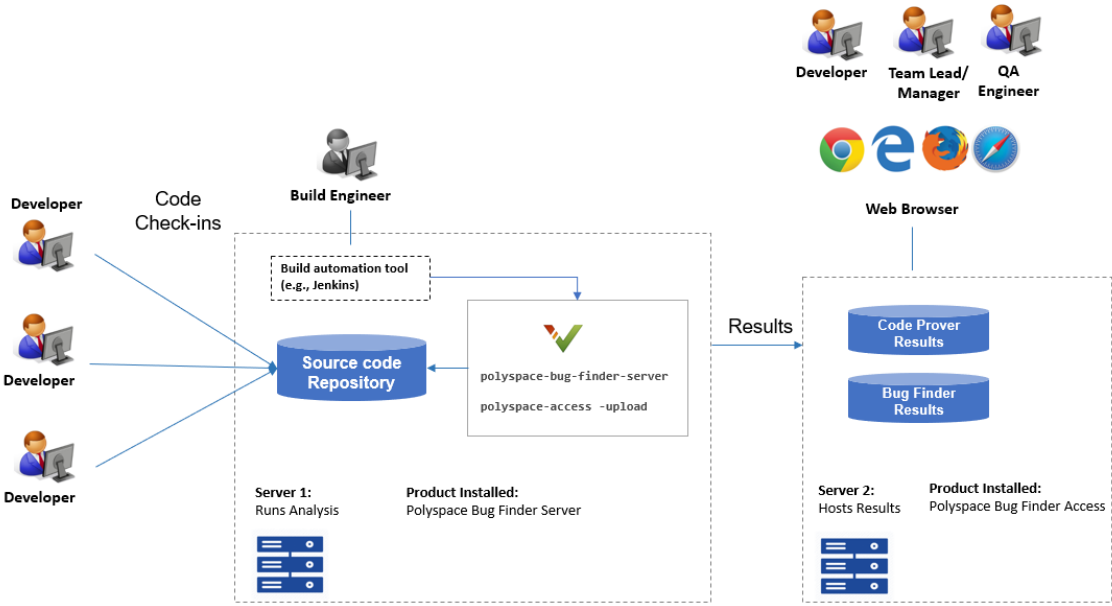
Benefits: You can run the Bug Finder analysis on a few powerful server class machines but view the analysis results from many terminals.

With the desktop product, Polyspace Bug Finder, you have to run the analysis and view the results on the same machine. To view the results on a different machine, you need a

second instance of a desktop product. The desktop products can now be used by individual developers on their desktops prior to code submission and the server products used after code submission. See Polyspace Products for Code Analysis and Verification.

Continuous Integration Support: Run Bug Finder on server class computers with continuous upload to Polyspace Access web interface

Summary: In R2019a, you can check for bugs, coding standard violations and other issues on server class machines as part of continuous integration. When developers submit code to a shared repository, a build automation tool such as Jenkins can perform the checks using the new Polyspace Bug Finder Server product. The analysis results can be uploaded to the Polyspace Access web interface for review. Each reviewer with a Polyspace Bug Finder Access license can login to the Polyspace Access web interface and review the results.



Note: Depending on the specifications, the same computer can serve as both Server 1 and Server 2.

See:

- Install Polyspace Server and Access Products
- Run Polyspace Bug Finder on Server and Upload Results to Web Interface

Benefits:

- *Automated post-submission checks*: In a continuous integration process, build scripts run automatically on new code submissions before integration with a code base. With the new product Polyspace Bug Finder Server, a Bug Finder analysis can be included in this build process. The analysis can run a specific set of Bug Finder checkers on the new code submissions and report the results. The results can be reviewed in the Polyspace Access web interface with a Polyspace Bug Finder Access license.
- *Collaborative review*: The analysis results can be uploaded to the Polyspace Access web interface for collaborative review. For instance:
  - Each quality assurance engineer with a Polyspace Bug Finder Access license can review the Bug Finder results on a project and assign issues to developers for fixing.
  - Each development team manager with a Polyspace Bug Finder Access license can see an overview of Bug Finder results for all projects managed by the team (and also drill down to details if necessary).

  For further details, see the release notes of Polyspace Bug Finder Access .

Continuous Integration Support: Set up testing criteria based on Bug Finder static analysis results

Summary: In R2019a, you can run Bug Finder on server class machines as part of unit and integration testing. You can define and set up testing criteria based on Bug Finder static analysis results.

For instance, you can set up the criteria that new code submissions must have zero high-impact defects before integration with a code base. Any submission with high-impact defects can cause a test failure and require code fixes.

See:

- `polyspace-bug-finder-server` for how to run Bug Finder on servers.
- `polyspace-access` for how to export Bug Finder results for comparison against predefined testing criteria.

If you use Jenkins for build automation, you can use the Polyspace plugin. The plugin provides helper functions to filter results based on predefined criteria. See Sample Scripts for Polyspace Analysis with Jenkins.

Benefits:

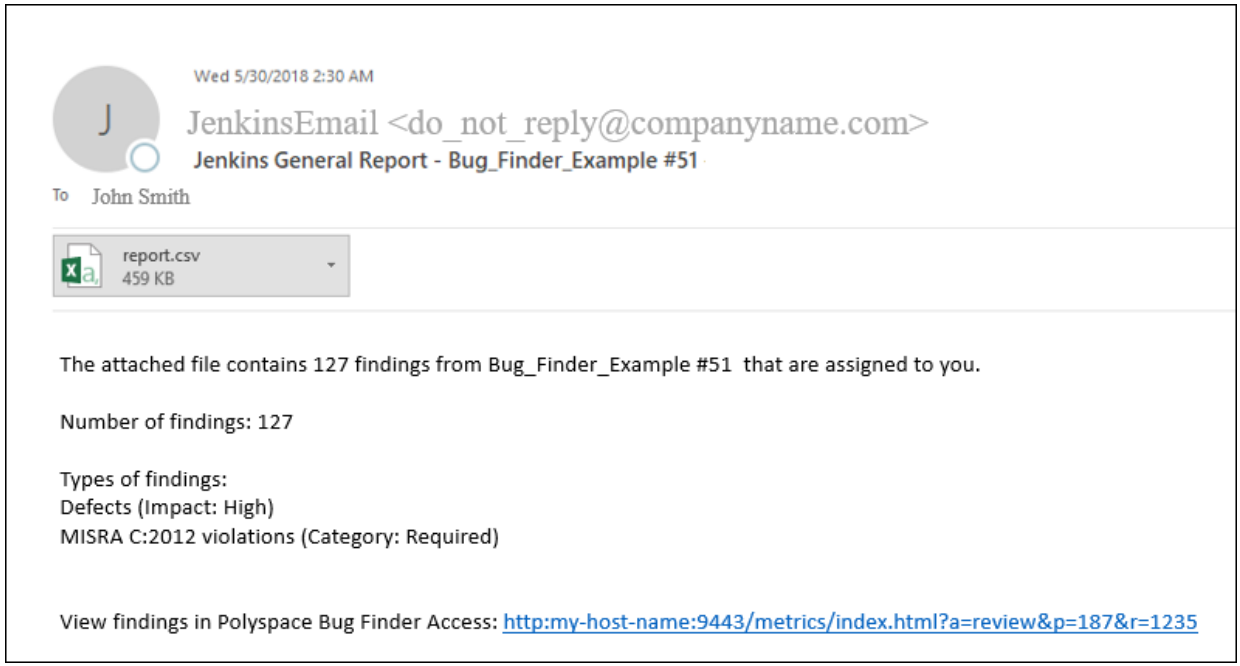
- *Automated testing*: After you define testing criteria based on Bug Finder results, you can run the tests along with regular dynamic tests. The tests can run on a periodic schedule or based on predefined triggers.
- *Prequalification with Polyspace desktop products*: Prior to code submission, to avoid test failures, developers can perform a pre-submit analysis on their code with the same criteria as the server-side analysis. Using an installation of the desktop product, Polyspace Bug Finder, developers can emulate the server-side analysis on their desktops and review the results in the user interface of the desktop product. For more information on the complete suite of Polyspace products available for deployment in a software development workflow, see Polyspace Products for Code Analysis and Verification.

  To save processing power on the desktop, the analysis can also be offloaded to a server and only the results reviewed on the desktop. See Install Products for Submitting Polyspace Analysis from Desktops to Remote Server.

Continuous Integration Support: Set up email notification with summary of Bug Finder results after analysis

Summary: In R2019a, you can set up email notification for new Bug Finder results. The email can contain:

- A summary of new results from the latest Bug Finder run only for specific files or modules.
- An attachment with a full list of the new results. Each result has an associated link to the Polyspace Access web interface for more detailed information.



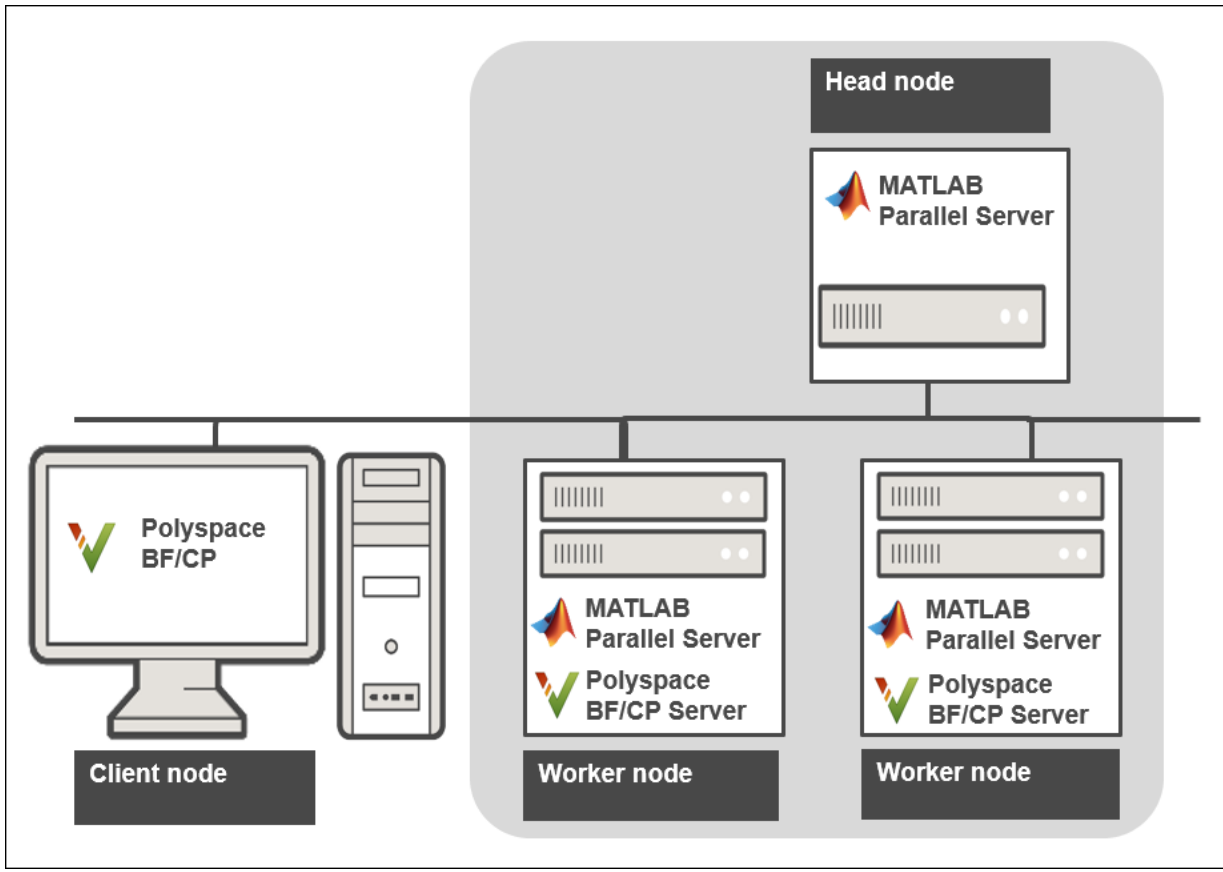
See Send E-mail Notifications with Polyspace Bug Finder Results.

Benefits:

- *Automated notification*: Developers can get notified in their e-mail inbox about results from the last Bug Finder run on their submissions.
- *Preview of Bug Finder results*: Developers can see a preview of the new Bug Finder results. Based on their criteria for reviewing results, this preview can help them decide whether they want to see further details of the results.
- *Easy navigation from e-mail summary to Polyspace Access web interface*: Each developer with a Polyspace Bug Finder Access license can use the links in the e-mail attachments to see further details of a result in the Polyspace Access web interface.

Offloading Polyspace Analysis to Servers: Use Polyspace desktop products on client side and server products on server side

Summary: In R2019a, you can offload a Polyspace analysis from your desktop to remote servers by installing the Polyspace desktop products on the client side and the Polyspace server products on the server side. After analysis, the results are downloaded to the client side for review. You must also install MATLAB® Parallel Server™ on the server side to manage submissions from multiple client desktops.



See Install Products for Submitting Polyspace Analysis from Desktops to Remote Server.

Benefits: The Polyspace desktop products have a graphical user interface. You can configure options in the user interface with assistance from features such as auto-population of option arguments and contextual help. To save processing time on your desktop, you can then offload the analysis to remote servers.