Testing BLUESPAWN

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

William Mayes
Fall 2021, Spring 2022

Technical Project Team Members
Will Mayes

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature Will Mayes Date 5/5/2022
Will Mayes

Approved yonghwi kwon Date 5/12/2022
Yongwhi Kwon, Department of Computer Science

Introduction

BLUESPAWN is an open-source active defense and EDR tool (Smith & Krist, 2020) for Windows computers developed by the UVA cyber defense windows team. The project's goal is to provide a tool for cybersecurity professionals to respond to advanced cyber threats. Endpoint Detection & Response (EDR) platforms are the state-of-the-art antivirus tools but come with the downside that they function largely as black boxes. BLUESPAWN provides insight into detections by identifying malware behaviors as defined by the MITRE (2019) ATT&CK Framework. BLUESPAWN is gaining attention from the cybersecurity community, having received over 900 stars on GitHub, but is not yet ready for at scale use. BLUESPAWN can be found at https://github.com/ION28/BLUESPAWN (Smith et al, 2020).

For any software to be usable at large scale, developers must extensively test it. This project is the first effort in including tests that will enable BLUESPAWN to be ready to deploy in production environments.

BLUESPAWN's Architecture

Jack McDowell (2021) provided an in-depth explanation of the design and architecture of BLUESPAWN. Understanding these decisions is essential to designing tests.

BLUESPAWN has four modes in which it can be run: hunt, mitigate, scan, and monitor. Hunt mode performs a point-in-time full system hunt for malware techniques mapping to the MITRE ATT&CK framework. Mitigate applies security settings according to input files that describe the mitigations. Scan performs a point-in-time scan on a specific file, registry key, or other windows artifact. Monitor watches the system for changes that could be associated with malware according to the MITRE ATT&CK framework and performs relevant hunts when changes are detected. Of these modes, hunt is the core of what BLUESPAWN attempts to

accomplish and is the most complete mode of all of them. For this reason, tests designed for this effort focus fully on this mode. Hunt mode both logs what it finds and attempts to remediate anything it finds through reactions. For hunt mode to be considered working, both logging and reactions must be working.

Methodology

Types of Testing

There are many ways to test production software. Unit tests test a single function or small functionality in a program to ensure it works as expected for when it's used in the larger project. Integration tests, also known as system tests, test the program in the context of the system in which it is running. Fuzzing is a type of testing that provides many random inputs to the program in the hope of catching any inputs that can cause a crash or hang. There are other forms of testing that can be beneficial to an enterprise project, but these were the 3 types of testing I considered for this project.

I determined fuzzing, while beneficial, was the least pressing form of testing to implement, as the form of errors it catches is rather narrow compared to integration and unit tests. To prioritize between the other testing types I discussed with the other project authors. We determined both were beneficial, but that we wished to reorganize some of the backend code to BLUESPAWN. Such reorganization would change the form unit tests would need to take, as a different set of functions would be implemented. It would not however, change the form of system tests, as the input and output of BLUESPAWN would remain unchanged. For this reason, I determined integration tests would be the focus of this project, with other tests being left for future work.

Test Design

Since the hunt mode in BLUESPAWN maps results to MITRE ATT&CK techniques, I decided that my integration tests should be mapped in the same way. I took inspiration for this approach, and for several of the written tests, from the Atomic Red Team project (Red Canary Co 2017). Atomic Red Team is a series of tests designed to map to the MITRE ATT&CK framework. Originally, these tests were integrated into testing for BLUESPAWN; however, they had several issues for this. First, the tests weren't designed with BLUESPAWN's detections in mind. Atomic Red Team tests are designed to test coverage of a certain scheme against the MITRE ATT&CK framework. Integration tests should be designed to ensure things that should be covered by the code that is written, are indeed covered. Second, Atomic Red Team tests didn't easily have a way to integrate into an automated pass or fail check for BLUESPAWN that would be useful in continuous integration. For these reasons, Atomic Red Team was used primarily for inspiration, but is not used in the integration tests themselves.

I split the integration tests into four phases: attack, hunt, check, and clean. Attack is responsible for performing malware-like activities on a system that corresponds to the given test. Hunt runs BLUESPAWN in hunt mode with the highest level of aggressiveness and remove-value and delete-file reactions. The check phase has two subphases: check caught and check fixed. Check caught searches BLUESPAWN's logs to ensure that every attack was caught. Check fixed searches the system to ensure that attacks were properly remediated if BLUESPAWN supports remediation. Finally, the clean phase undoes everything the attack phase did to ensure nothing is broken.

Each test must implement its own attack, check, and clean phases. Every test's attack phase is run, then one hunt phase is run for all of them, then every test's check phase is run, and finally every test's clean phase is run.

Attacks are written in either batch script or PowerShell. Attacks should be short and simple to do the minimum possible to create system artifacts that should be caught in a hunt. Most attacks are only one or two lines long and none include any complex logic. Clean phases are written the same way. Any registry keys or values, users, and files that the attack phase creates must be deleted by the clean phase. Any registry values that are edited by the attack phase must be set back to their default values by the clean phase to ensure that the system doesn't break from a necessary registry value being deleted.

The check fixed phase is written in PowerShell for every test. The script should confirm that all created "malicious" files and registry values have been deleted. Any hunts that have custom remediations, such as setting registry values back to default, should be tested to make sure that the custom remediation occurs rather than deletion. Every file must return true if the test passes and false if the test fails. If the test fails a failure message describing why the test failed is printed to the console. The check caught phase for every test consists of a JSON file. The JSON file describes the list of detections that should be present to pass. Each described detection has the name of the hunt that will be listed in the detection, the detection type, and additional information based off the type of detection. It also includes a string that should be printed in the case of a failed check. During the check fixed phase, each JSON file is read into a list and then the detections log is compared to every JSON object in the list to see if they match. If no match exists, the failed check string is printed out to the console which should provide context to what failed.

Results

I designed 51 integration tests for this project. All testing was done on a develop branch build of BLUESPAWN compiled on January 8, 2022. Of the 51 tests, 10 tests completed and failed, and 11 tests caused the execution of BLUESPAWN to hang indefinitely. Additionally, all tests that involved deleting a DLL file hung for a period of around two minutes. This was since the delete-file reaction attempts to unload the DLL from all running processes, but BLUESPAWN did not have access to every process. It improperly handled the lack of permissions to a process and hangs briefly instead of continuing. The 41% of tests that failed is an indicator of the need for this style of tests. A full list of tests can be found in the appendix.

Failing Tests

Of the 10 failing tests, seven failed due to missing detections, three failed due to failure to properly react, and one failed due to a bug in the logging of the detection. This sums to more than 10 because one test failed to detect one part of the attack and failed to remove the part it did detect, so it failed in two ways.

The 1 test that failed due to a bug in logging was a hunt for MITRE ATT&CK technique T1547, sub-technique 010 (abbreviated T1547.010). The sub-techniques in MITRE ATT&CK are all three digits long with leading zeros if necessary. As such, we record them like that as well. However, in the translation from the C code which labels it as 010 to the JSON which logs it, the three-digit number is treated like a C literal. In C, leading with a zero in an integer literal denotes that the number is represented in octal. When that number then gets translated to JSON, it is treated as the number 10 base eight, then gets logged as sub-technique 008.

The seven tests that failed due to missing detections had several different reasons for failing. Two tests for T1547.002 and one for T1547.005 failed due to use of an internal BLUESPAWN function called SearchPathExecutable. This function is intended to find an executable file in the search path of the computer and is used for registry values that don't include the whole path to the file. However, in these two sub-techniques, the value it is searching for is a DLL file, and the implementation of SearchPathExecutable only properly works when searching for EXE files. This is an error that could be caught with the implementation of unit tests. A test for T1547.003 fails as the hunt does not register that the detection it finds in the registry references a file. As a result, it fails to ever detect the malicious file. A test for 1569.002 fails in a similar manner. This test sets a registry key to be equal to the value "cmd.exe /c malicious.dll". BLUESPAWN fails to find the associated malicious file that is referenced in the command. Tests for T1547.004 and T1553.003 both fail due to detection certainties being improperly too low, causing them to never be logged.

The three tests that failed for failing to properly react were for T1547.003, T1553.003, and T1569.002. It was unclear in testing and from the error logging what caused these tests to fail. No efforts were made in the attacks to protect the registry values and files from being deleted or edited, so no reactions should ever fail for any reason.

Hanging Tests

The hanging tests were all the tests for MITRE ATT&CK techniques T1505 and T1546, as well as a test for sub-technique 002 of technique T1068. All hanging tests resulted in BLUESPAWN printing out a generic error message after which it hung and never completed. It was allowed to run for around 15 minutes for all 3 failing tests before it was determined that it

would not continue execution. In actual testing, these tests have to be skipped, as the hanging prevents the other tests from completing.

Conclusion

I designed and implemented 51 integration tests for the BLUESPAWN windows defense tool. These tests were designed to ensure that the overall execution of the BLUESPAWN hunt mode correctly finds and remediates all attacks it says it should. The tests resulted in 21 failing tests: 10 due to improper execution and 11 due to hanging. These tests provide a baseline for ensuring that BLUESPAWN executes without bugs and can be used in enterprise environments. They also provided information that can be used to fix bugs currently present in BLUESPAWN. These tests are a great start to implementing a full suite of testing that can enable BLUESPAWN to become an enterprise ready tool.

Future work on testing BLUESPAWN should prioritize adding unit tests. Unit tests can help to prevent some of the errors that were found by the system tests from happening in the future as well as identifying additional bugs. Both unit tests and integration tests should be added to the continuous integration pipeline and guidelines should be implemented to instruct that these tests be implemented for all new functions and hunts. Further integration tests should be added for the other modes of BLUESPAWN to ensure it works in all its intended uses. Finally, non-pipeline testing like fuzzing and experimental testing against actual malware to further provide confidence and a measure for how BLUESPAWN performs.

References

McDowell, J. (2021, May 17). *BLUESPAWN Design and Architecture*.

MITRE Corporation. 2019. Matrix - Enterprise | MITRE ATT&CK®. MITRE ATT&CK. *https://attack.mitre.org/matrices/enterprise/windows/*

Red Canary Co. 2017. Atomic Red Team. https://github.com/redcanaryco/atomic-red-team/

Smith, J., Krist, C. (2020, May 8). *BLUESPAWN: An Open-Source, Active Defense & Endpoint Detection and Response (EDR) Software for Windows-based Systems*.

Smith, J., Krist, C., Mayes, W., & McDowell, J. 2020. BLUESPAWN. *https://github.com/ION28/BLUESPAWN*

Appendix

Table of tests

Test (Technique.sub.number)	Description	Result
T1037.001.001	Adds a UserInitLogonScript DLL in the HKCU\Environment key and adds the DLL to C:\Windows\Temp	Passes
T1053.005.001	Adds a scheduled task that runs a mock malicious EXE and places the EXE in C:\Windows\Temp	Passes
T1068.XXX.001	Adds a DLL to the printer\ports registry value and adds the DLL to C:\Windows\Temp	Passes
T1068.XXX.002	Adds a DLL to the current version\ports registry value and adds the DLL to C:\Windows\Temp	Hangs
T1070.006.001	Adds an EXE to C:\Windows\Temp and then timestomps it	Passes
T1136.001.001	Adds a user with username $ (which is hidden from the net user command)	Passes
T1484.XXX.001	Adds a file called ntuser.man (related to GPO typically) to the current user's folder	Passes
T1505.003.001	Adds a PHP webshell to the C:\inetpub\www directory	Hangs
T1543.003.001	Adds a ServerLevelPlugin value in the DNS service registry key to reference a DLL that is added to C:\Windows\Temp	Passes
T1546.002.001	Adds a screensaver that references a malicious EXE that is added to C:\Temp	Hangs
T1546.007.001	Adds a malicious registry value to the netsh key that references a DLL that is added to C:\Windows\System32	Hangs
T1546.008.001	Adds a sticky keys backdoor that references an EXE added to C:\Temp	Hangs
T1546.009.001	Adds a malicious AppCertDLL and places it in the C:\Temp folder	Hangs
T1546.010.001	Adds a malicious AppInitDLL and places it in the C:\Temp folder	Hangs
T1546.010.002	Adds a malicious AppInitDLL in the WOW6432 node registry key and places it in the C:\Temp folder	Hangs
T1546.011.001	Adds a fake malicious shim and places the database in C:\Program Files	Hangs
T1546.012.001	Adds a malicious program to monitor notepad.exe in the registry and places the file in C:\Temp	Hangs
T1546.015.001	Adds a fake, malicious CLSID and places the referenced DLL in C:\Temp	Hangs
T1547.001.001	Adds a malicious run key in Current Version\Run and adds the referenced	Passes

	EXE to C:\Windows\Temp	
T1547.001.002	Adds a malicious run key in Current Version\RunOnce and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.003	Adds a malicious run key in Current Version\RunServices and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.004	Adds a malicious run key in Current Version\RunOnceServices and adds the references EXE to C:\Windows\Temp	Passes
T1547.001.005	Adds a malicious run key in Current Version\RunOnceEx and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.006	Adds a malicious run key in Current Version\RunOnceServicesEx and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.007	Adds a malicious run key in Explorer\Run and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.008	Adds a malicious run key in Command Processor\AutoRun and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.009	Adds a malicious startup value in Explorer\User Shell Folders and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.010	Adds a malicious common startup value in Explorer\ Shell Folders and adds the referenced EXE to C:\Windows\Temp	Passes
T1547.001.011	Adds a known malicious CLSID and places the DLL in the known location	Passes
T1547.002.001	Adds a malicious LSASS authentication package DLL and places the DLL in C:\Windows\Temp	Fails due to an issue with SearchPathExecutable not working with dlls
T1547.002.002	Adds a malicious LSA extension DLL and places the DLL in C:\Windows\Temp	Fails due to an issue with SearchPathExecutable not working with dlls
T1547.003.001	Adds a malicious time provider and places the DLL in C:\Windows\Temp	Fails to register the registry value as a file reference so it doesn't find the file. Also fails to remove the registry value.
T1547.004.001	Adds a malicious UserInit value in Current Version\WinLogon and adds the DLL to C:\Windows\Temp	Passes
T1547.004.002	Adds a malicious Shell value in Current Version\WinLogon and adds the DLL to C:\Windows\Temp	Passes
T1547.004.003	Adds a malicious UserInit value in Current Version\WinLogon and adds the EXE to C:\Windows\Temp	Passes
T1547.004.004	Adds a malicious WinLogon\Notify value	Fails due to erroneously low

	and adds the DLL to C:\Windows\Temp	certainty
T1547.005.001	Adds a malicious LSASS security package and places the DLL in C:\Windows\System32	Fails due to an issue with SearchPathExecutable not working with dlls
T1547.010.001	Adds a malicious print driver and places the DLL in C:\Windows\System32	Fails due to an error in logging causing 010 to be logged as 008.
T1548.002.001	Places a malicious EXE in the DelegateExecute key and places the EXE in C:\Windows\Temp	Passes
T1548.002.002	Adds a malicious value to the mscfile\shell\open\command key and places the EXE in C:\Windows\Temp	Passes
T1553.003.001	Adds a malicious Crypt SIP DLL and places the DLL in C:\Windows\Temp	Fails to remove the malicious file
T1553.003.002	Adds a malicious Crypt SIP Function to the registry.	Passes
T1553.003.003	Adds a malicious Crypt SIP DLL and function to the registry	Passes
T1553.003.004	Adds a malicious signature provider DLL	Fails to record the detection due to low certainty
T1553.003.005	Adds a malicious signature provider function	Passes
T1553.003.006	Adds a malicious signature provider DLL and function	Passes
T1562.004.001	Adds a malicious authorized application to the firewall registry and places the EXE in C:\Windows\Temp	Passes
T1562.004.002	Adds a malicious globally open port to the firewall	Passes
T1569.002.001	Adds a malicious service and places the referenced EXE in C:\Windows\Temp	Passes
T1569.002.002	Adds a malicious service with command “cmd.exe /c malicious.exe” and places the malicious EXE in C:\Windows\Temp	Fails to find the file referenced after the /c
T1569.002.003	Adds a malicious service and a malicious service DLL then places the DLL in C:\Windows\Temp	Fails to remove the registry value creating the malicious service DLL