Catch Your Own Bugs: Including all Engineers in the Automation Cycle

Laura Bright
Laura_bright@mcafee.com

Abstract

End-to-end automation provides many benefits to developers and QA engineers including faster defect detection and greater product stability. Automation reduces testing overhead and continually monitors overall product quality, enabling developers to quickly identify and fix defects in both new features and regressions.

A challenge to the success of any automation framework is involving the engineers who benefit from this effort. Our experience from past projects has been that most engineers do not monitor automation runs closely due to time constraints or limited accessibility of results. Automation team members are left with the task of interpreting results and filing defects, which places a greater load on the automation team and increases the overall time required to find and fix defects.

This paper presents solutions we have introduced to close this gap in the automation cycle and involve all engineers in the automation process. We draw on experience building end-to-end automation frameworks for two projects and present specific examples of how our solutions have helped developers catch and fix their defects in a timely manner. Our solutions include the following:

- Automatic automation triggers - every time a new build is ready, the automated tests begin executing immediately.
- Email notification of results - An automatic email summary of test results is sent to all team members after every run, ensuring visibility and a prompt response.
- User-friendly UI for result reporting and analysis - Engineers do not need detailed knowledge of the underlying framework implementation to view results and identify the root cause of a defect. Once a defect is fixed and a new build is available, the automation suite will execute automatically and verify the fix.
- All engineers actively contribute to the framework through test script authoring, feedback, and maintenance.

To date, our automation has helped developers and QA catch 19% of the total defects in our current project. Our daily automation runs and reporting solutions have greatly increased developer and QA involvement in automation efforts and improved the stability of our weekly builds.

Biography

Laura Bright is a Software QA Engineer at McAfee where she has led automation framework development efforts for several endpoint security products. She has an A.B. degree from Dartmouth College and a Ph.D. degree in Computer Science from the University of Maryland College Park.

1 Introduction

An end-to-end automation framework provides real-time product quality monitoring and ensures that new issues and regressions are detected in a timely manner. Automatically executing a suite of functional tests on every new build provides immediate feedback on the effects of changes to the code and determines if a build is suitable for further testing.

A challenge to the success of such a framework is enabling all stakeholders to promptly interpret test results and respond to issues. Handling new issues requires sufficient information to determine if a failure is caused by a product defect, and if so, file the defect so it can be fixed as soon as possible. A summary of which tests passed or failed provides a high-level overview of the health of the product, but does not provide sufficient information to identify the cause of failures. If the automation results are not easily accessible and understandable by all team members, team members must rely on automation engineers to interpret test failures and file defects and the cycle is not fully automated. Further, all team members cannot be expected to continually monitor automation results, and should be notified automatically when new results are available.

This paper presents efforts by the McAfee Endpoint Security Automation team to close this gap in the automation cycle and enable developers to catch their own defects.

Our specific goals include the following:

- **Keep all team members in the loop** – All developers, QA engineers, and managers involved with the project should always be aware of the latest automation results.
- **Minimize overhead for automation team** – All stakeholders should be able to drill down and analyze test results without requiring help from an automation engineer.
- **Ease of writing, maintaining, and understanding test scripts** – All developers and QA engineers should be familiar with the automation scripting language and process. This not only allows everyone on the team to contribute to automation efforts, but also improves their understanding of how scripts work so they can easily troubleshoot issues.

We present our experiences extending the automation framework and processes to increase the involvement of all developers and QA engineers in monitoring and interpreting automation results. We implemented new features in our existing automation framework to improve the effectiveness of the automation result reporting and provide detailed information about every test executed. We also discuss increasing involvement of both developers and black box engineers in the script writing process and show how a tool that is easy to learn has many uses beyond the existing automation framework.

Section 2 presents an overview of the team, process, and product, as well the limitations of previous automation frameworks. In Section 3 we provide an overview of the automation framework and present the features that were added to help developers find and fix their own defects. Section 4 presents results including examples of increased developer involvement and defects that were fixed early, and Section 5 discusses future directions and concludes.

2 Background

This section presents an overview of the product, team, and test cases for the automation project presented in this paper.

2.1 Products and Team Structure

The McAfee Endpoint Security Automation team handles test automation for several anti-virus and firewall products. Previously the team built an automation framework for the MOVE (McAfee Optimized for Virtual Environments) product. The current automation effort is testing the latest release of the enterprise endpoint security product. The product has several anti-virus and firewall features including:

- On-access scanner
- On-demand scanner
- Outlook and Lotus Notes Scanners

The development and QA teams are evenly distributed between offices in Beaverton and Bangalore, with automation engineers on the QA teams in both locations. An automation rig is maintained at each location and all required platforms and configurations are tested at one of the two sites. Due to the geographic distribution of the teams, code changes occur 24 hours a day and it is important to catch defects as soon as possible. If an engineer at one site introduces a defect and does not detect it before the end of the day, team members at the other site may need to wait a full day before the engineer fixes the defect if they are unable to fix it themselves and the productivity of the entire team is impacted.

2.2 Build Process

A central build server automatically generates one new build per day. In addition, any team member can request a new build at any time, for example to test new changes that have been checked in. Every build has its status recorded in a central database to identify its status. At least one build per week is marked as “Released to QA” (RTQA) and is used by the entire QA team to execute their test suites. It is important to have a stable RTQA build, otherwise QA team members may not be able to execute their tests in a timely manner.

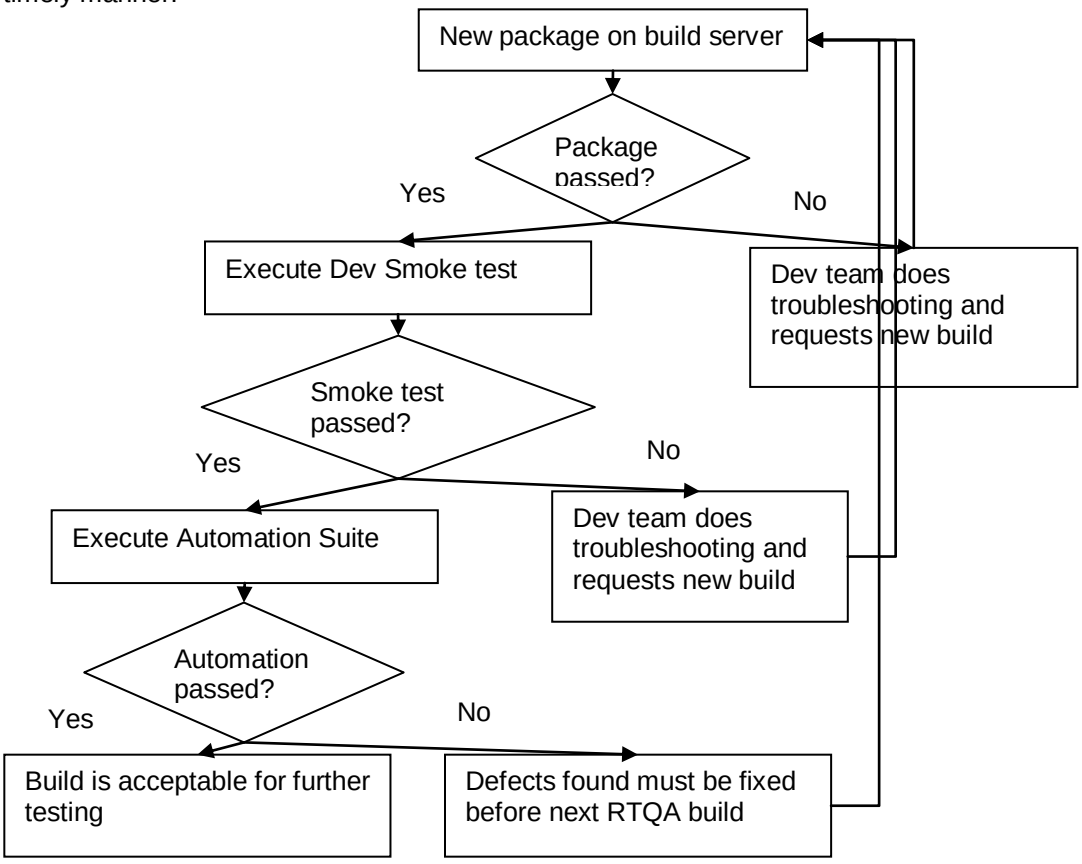

Figure 1: Build Process Flowchart

The build process is outlined in Figure 1. An automatic smoke test created by the development team launches on every successful package to verify basic product functionality. If the smoke test passes, the status in the build server database is updated to “Dev Smoke Test Passed” and the build is ready for additional automated testing. The end-to-end automation rig will automatically begin executing tests on every build that passes the smoke test. If the automation run passes, the build is acceptable for further

testing by the entire QA team. If the automation run fails, any defects that are caught are expected to be fixed no later than in the next RTQA build, sooner if the defect blocks other QA tasks.

2.3 Test Suites

The Endpoint project includes over 4300 test cases, more than 1200 of which are automatable. All tests were audited by members of the automation team to identify whether or not a test case is automatable and the relative automation priority of tests. Automated tests are divided into Build Verification Test (BVT) and Functional Verification Test (FVT). BVT tests cover basic functionality and execute automatically on every nightly build. FVT tests cover product functionality in greater depth and execute on every RTQA build.

2.4 Challenges

Several challenges drive our efforts to improve our automation framework and include all stakeholders in the automation cycle:

- **Frequent code changes and builds** – During an iteration there is a high degree of code churn and significant changes to the code may occur on a daily basis. There is at least one build per day and often several builds if developers are checking in and testing their changes. Manual black box testers cannot keep up with this build frequency, however it is important to catch new and regression defects early so they can be resolved before the next RTQA build.

- **Dependencies on other products** – The Endpoint product consumes and interacts with several other McAfee products, for example the anti-virus engine. Each product has its own QA team and automation setup to monitor product quality, and new milestone builds of dependent products are automatically added to Endpoint product builds when they become available. However, even if a dependent product passes its own tests, changes to the product may impact features of the Endpoint product, or features of the Endpoint product may break due to a previously unknown defect in the dependent product. In addition to detecting defects due to changes in our own product, it is important to detect defects in dependent products so they can be addressed in a timely manner.

- **Geographically distributed team** – With two sites on opposite sides of the globe, product development and testing efforts are continuous. The start of the work day in Beaverton coincides with the end of the work day in Bangalore, and vice versa. This arrangement ensures nearly 24-hour coverage, but has the disadvantage that team members at one site need to wait a day if they need assistance from someone at the other site. For example, if a developer at the Beaverton site checks in code that breaks a product feature, the team in Bangalore may identify a defect but will need to wait until the next day until it is fixed if they are unable to fix it themselves. To maximize productivity, it is important to find and fix defects as soon as possible so team members at the other site will not be impacted.

2.5 Limitations of Earlier Automation Efforts

Our past experience on previous product releases has shown that automation results are not closely monitored by most team members outside of automation. One reason for limited stakeholder involvement is lack of time. Automation runs execute at least once per day, and most stakeholders do not have the time to review results on the web server regularly. Also, most will not think of checking the server unless they receive regular reminders or are monitoring a specific issue.

A second reason is limited access to troubleshooting information and difficulty of interpreting the information. In our previous projects, a web server with the number of tests that passed and failed on every run was available to all stakeholders. While this web server provides a good overview of product

health, it does not provide sufficient information to troubleshoot the root cause of failures. Selected log files from every automation run were made available on a file share, but interpreting these files often required detailed knowledge of the automation framework.

As a result of these limitations, developers and black box engineers typically relied on an automation team member for the status of the latest run rather than monitoring results themselves. Automation engineers had to sift through the logs to identify the root cause of issues and file defects. This process placed an undue burden on automation engineers and increased the length of time required to troubleshoot defects. It also did not fully maximize the benefits of end-to-end automation since intervention from automation engineers was needed. Thus, a major goal of our current automation efforts was to extend the framework to provide detailed information from all automation runs, and provide regular summaries to all team members about the status of current automation runs.

3 Automation Framework Overview

This section presents the architecture of the automation framework and describes specific features we implemented to make the framework accessible to all team members.

3.1 End-to-end Automation

Figure 2 shows the architecture of the end-to-end automation framework. A script that monitors the build server executes continuously on the automation controller. This script queries the build server database every few minutes. Every time the script detects a new build in the database with a status of “Dev smoke

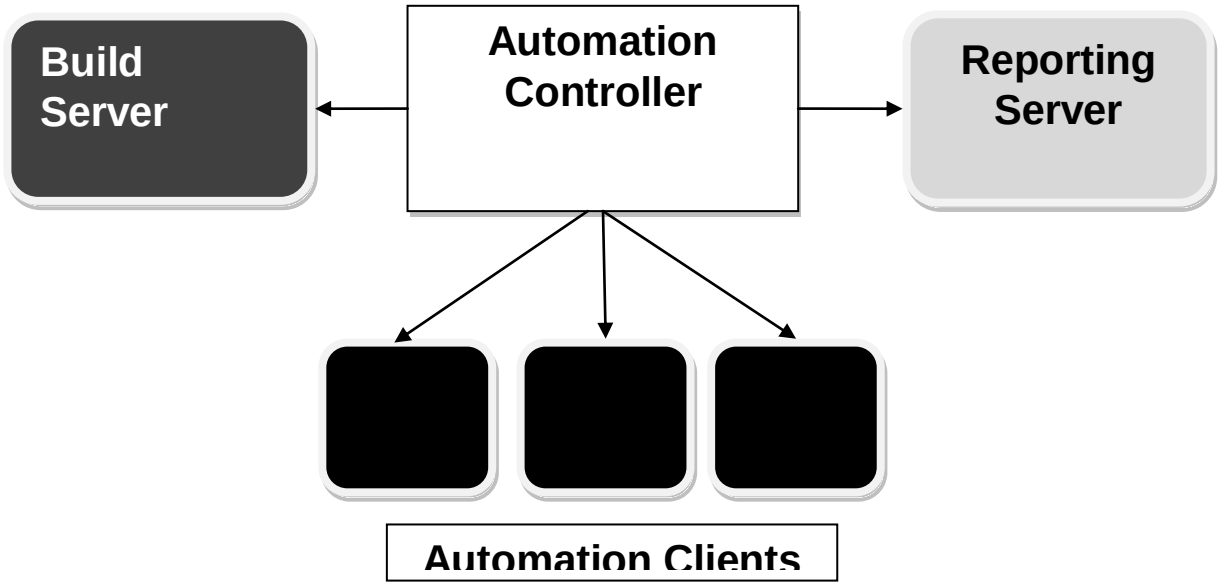

Figure 2:Automation framework architecture

test passed" it launches the automation run. The automation executes on a VMWare ESX server with several client images. Every time a new build is available, the automation controller reverts all clients to a clean snapshot, installs the new build on all clients, and launches an automation batch script on each client to execute all BVT test cases. As tests complete, their status is updated on a web server to indicate PASS or FAIL. When the batch run completes on each client, all log files are copied to the reporting server and user-friendly automation reports are generated on the reporting server. An automatic summary email is sent to all stakeholders with a list of failed tests and their components.

3.2 Test Scripts and Functions

The automation suite executed on each client consists of a batch script that runs a set of test scripts. The test scripting framework consists of a library of PERL functions that are called by individual test scripts. Each tests script is a text file with a sequence of function calls, and is parsed and executed by the framework. No detailed programming knowledge is required and any engineer can read and write test scripts. Our goal was to make the framework as easy to learn as possible and goes along with our efforts to incorporate all engineers in all stages of the automation cycle.

```
SetLogLocation ‘1234.txt’
CreateEicars ‘eicar.exe’
VerifyLogTextC ‘Deleted.*eicar.exe’ 1 ‘1234.txt’ 1
VerifyFileExists ‘eicar.exe’ 0
```

```
1 PASS Log location set to 1234.txt
2 PASS Eicar file eicar.exe successfully created
3 FAIL       Specified text ‘Delete.*eicar.exe’ not found
4 PASS File ‘eicar.exe’ does not exist. Expected.
```

Figure 3:Test script and log example

Each framework function call is a test step and reports PASS or FAIL to a log file along with the details of the test step. . If one or more steps fail then the test case fails. Users can review the automation logs to see which test steps failed and identify the cause of the failure. An example is shown in Figure 3. At the top is a simple test script that changes the product log name, creates a sample file to be detected by the on-access scanner, and verifies that the file was deleted by the product and the detection was logged. On line 3 of the sample automation log, the specified text is not found in the product log, so there may be a logging code defect.

In addition to improved understanding of test results, a benefit of an easy-to-use test script language is that all developers and QA engineers can contribute by writing test scripts as well. We discuss these efforts further in Section 4.2.

3.3 Email Notifications

Automatic email notifications are sent after every run and include a list of tests that failed on each platform as well as a summary of the number of test failures per component. The email also indicates clearly if the install failed on a particular operating system. Since the dev smoke test only covers a single platform and installation failures often apply to a particular class of operating system, the nightly runs are helpful to detect installation issues that may have been missed by the smoke test. An example email summary is shown in Figure 4.

Since the automation runs continuously, a run may complete during nighttime hours at one of the two sites and many automation engineers may be out of the office. Automatic emails ensure that stakeholders are immediately aware of automation results even if no automation engineers are present. In some cases,

developers can be notified of new issues and identify the root cause before any automation engineers have had a chance to look at the results. We present examples in Section 4.

All managers and team members at both sites receive these emails. Initially some team members did not closely monitor the emails, but over time nearly all team members have started to monitor them and respond promptly to unexpected failures.

```
Results for 2K3ER2:
        0 tests passed and 1 failed
       Failed tests:  19045  ***** NOTE: Install failed on 2K3ER2! *****
Results for 2K8R2:
        115 tests passed and 7 failed
       Failed tests:  11336 8415 8416 11016 7544 7531 9580
Results for Win7x86:
        129 tests passed and 8 failed
       Failed tests:  11336 8415 8416 7544 7531 7712 11373 9580
Results for XP3:
        125 tests passed and 12 failed
       Failed tests:  11336 8415 8416 7544 11016 10726 9304 7712 9640 9560 9793 9580

Failed tests by component:
AV-Outlook Email Scan=>OnDelivery ScanSettings: 1
AV-OnAccessScan=>Exclusions: 5
AV-OnAccessScan=>Process Classifications: 2
AV-OnDemandScan=>ScanSettings: 9
AV-Outlook Email Scan=>OnDemand ScanSettings: 5
VSE/Functional/AV/OnAccess Scan/Process Classifications: 1
AV-LotusNotes Email Scan=>OnAccess ScanSettings: 2
Common-Installer=>Default Install: 1
AV-ScriptScan=>Exclusions: 1
AV-Quarantine Manager=>Populate: 1
```

Figure 4: Example email summary

3.4 Reporting Web Server

The reporting web server contains detailed automation logs that indicate which steps of each test script passed and failed, as well as a graphical interface to summarize this information and allow users to drill down to individual results. Historical data is saved from earlier builds and iterations, so users can compare earlier runs against the current run. An example web report is shown in Figure 5. The drop-down menus at the top of the page allow users to select the build and iteration (the default is the most recent build). The automation summary shows the percentage of tests that failed on at least one platform, and the percentage of tests that passed on all platforms. If you click on an individual test, it displays the test results on each platform, and you can click on an individual platform to see which test steps failed as we will show in Section 4.

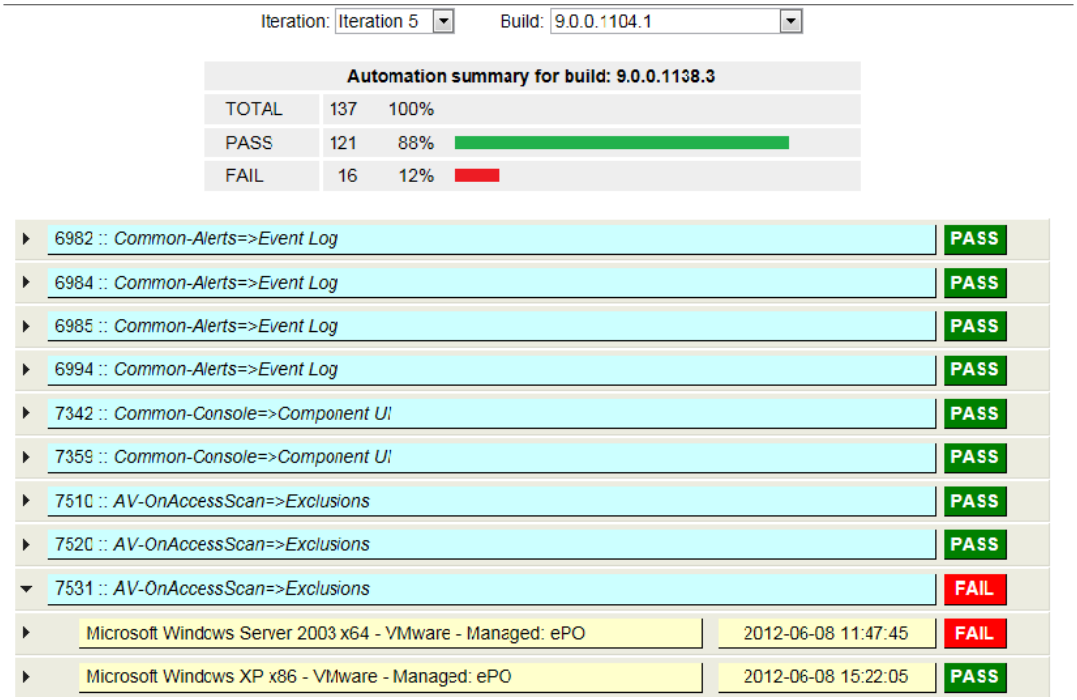

Figure 5: Automation report web server

The server also stores the following files from every automation run to help with troubleshooting. Access to these files is linked from the web reports so users can easily drill down for more information.

- **Product logs** – All logs from the product. This includes install logs which are helpful if the install fails. Logs are renamed in each test case to include the test id in the name so they can be easily identified.
- **Product configuration** – Each test saves a text file which contains the product settings at the time the test was run. If a test fails because a product feature did not perform as expected, these files are useful to verify that the product settings were as expected and determine if the failure is a product defect or test script error.
- **Event logs** – Windows event logs are saved and can be used to verify that product-related events are reported properly, or for troubleshooting if any unexpected system behavior occurs (e.g., a process crash).
- **Crash dumps** – If any process crashes during test execution, a WinDbg crash dump is automatically generated and copied to the report server at the end of the test run

3.5 Process

As a team we adopted the following practices for responding to test failures. Nightly BVT runs execute only on stable test scripts that are expected to pass, so automation failures are rare and failures normally indicate new or regression defects that should be filed and fixed in a timely manner. In some cases a developer can identify the root cause of the defect using the web server reports, and in these cases they are expected to fix the issue immediately even if a defect has not yet been filed. If assistance from an automation engineer is required, the engineer will analyze the failure and file a defect. If it is determined that automation framework logging could be improved to make troubleshooting easier, a defect will be filed against the automation framework and an automation engineer will make the fix. We present specific examples of defects that were found and fixed by developers in the next section.

4 Results

This section outlines success stories of our framework including specific examples of defects that were found and fixed by our framework. Some major benefits of the framework include:

- Earlier detection and fixing of issues – Automation runs occur 24 hours a day whenever a new build is ready, so results can be immediately reported to stakeholders. Several defects detected by nightly automation were fixed by developers before QA had time to file a defect. Many additional defects (>50) were observed immediately and prompted developers to ask the automation team about the status of the build. For example, there were 21 installation defects that indicated prompt action was needed. The automation runs also provided a good status monitor and provided automatic verification of bugs that were fixed.
- Information from all runs is stored on a server accessible to entire team. Historical information is available to monitor product quality over time.
- Reduced time in testing cycle and increased testing frequency
- Improved product stability – defects detected in nightly runs are fixed before the next RTQA, leading to better quality of RTQA builds and lower probability that an RTQA build will be rejected.
- Increased defect detection – There have been 19% of total defects found early through automation. Many of these defects were crashes that were difficult to reproduce in a manual environment, and 51% of Severity 1 defects were found through automation.

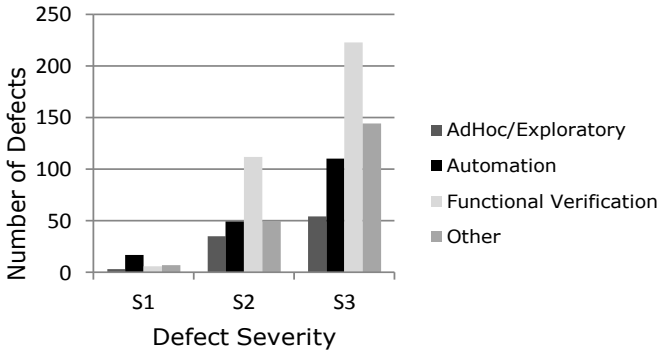

Figure 6: Defect severity by methodology

	S1	S2	S3	S4	Total
Percent defects found	51%	20%	20%	10%	19%

4.1 Success stories

This section presents a few examples of new defects and regressions that were rapidly detected by the nightly BVT automation and reporting framework. The nightly automation suite contains only tests that pass reliably, so any failures are considered regressions that need to be addressed promptly.

4.1.1 On-Demand Scan Regression

A developer in Beaverton made some changes to the configuration of which folders to scan in on-demand scan tasks. After these changes were checked in, three on-demand scan tests that had been passing previously started failing on the next build. The developer who had made the change observed that the failed tests were related to the on-demand scanner so they were likely related to the changes.
The web report for one of the three failed tests is shown in Figure 7. This test script configures an on-demand scan on a mapped network drive, and verifies that the samples on the drive are deleted and the detections are logged. In this example, the log indicates that the sample files were not logged or deleted, indicating that the on-demand scan did not scan the proper location. The developer promptly corrected the issue.

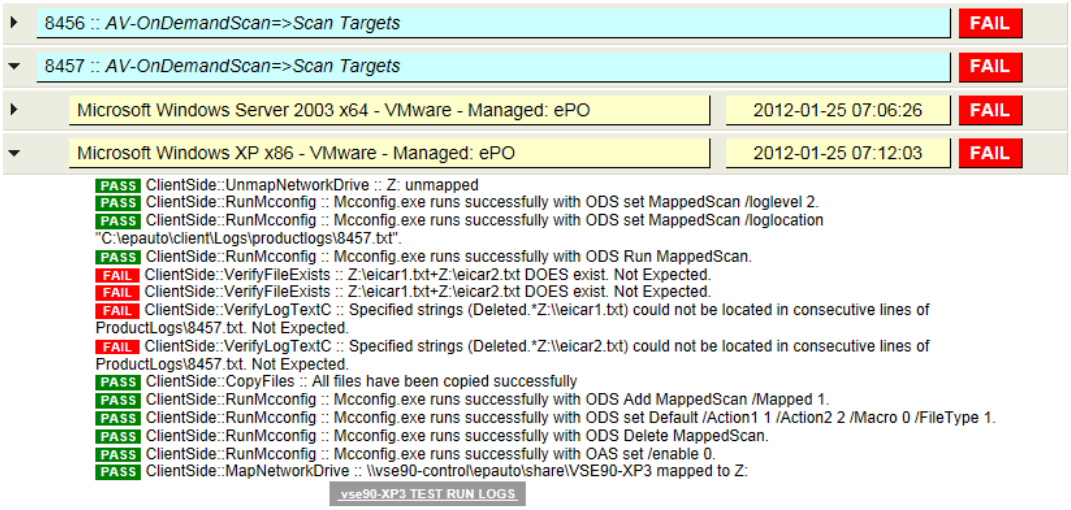

Figure 7: On-demand scan report indicating failure to scan mapped drive

4.1.2 Lotus Notes Scanner Regression

A developer in Bangalore checked in some changes to the Lotus Notes scanner code and requested a new build on the build server. The automation rig in Beaverton automatically launched the BVT suite on the new build, even though it was nighttime in Beaverton and no engineers were present. After the run completed, an automated email summary was sent to the entire team indicating that all of the Lotus Notes scanner tests had failed.

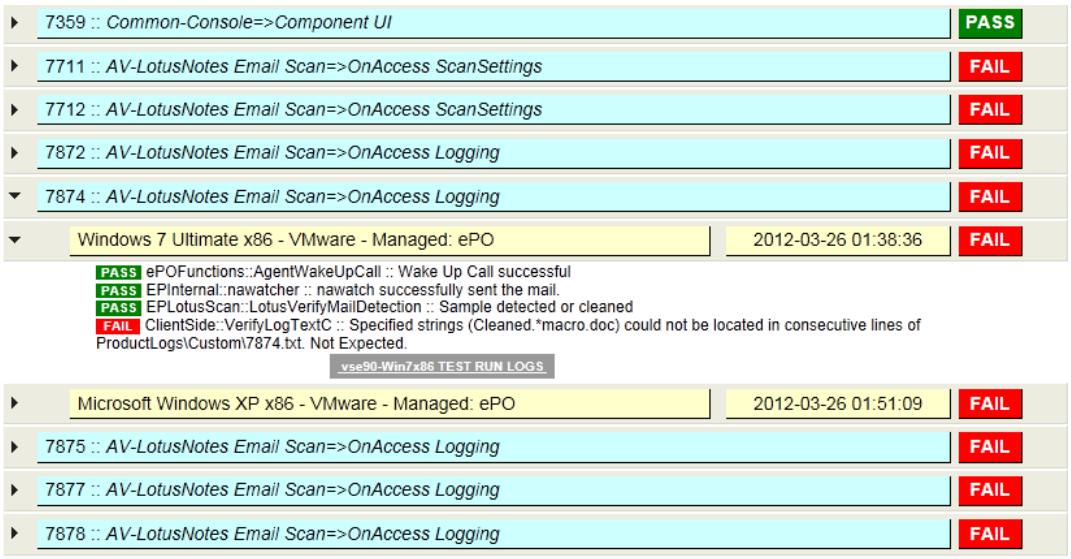

Figure 8: Web server report indicating Lotus Notes Scanner logging failures.

The developer who had checked in the changes immediately determined from the email message that the changes had broken some of the Lotus Notes Scanner functionality. Further investigation of the web server results indicated that the logging functionality was not working as expected (Figure 8). This log shows that the LotusVerifyMailDetection function passed, indicating that the sample was detected by the scanner, but the VerifyLogTextC failure indicates that the detection was not properly logged. The developer promptly notified the team that he was aware of the issue and was working to fix it. When automation engineers arrived at work in Beaverton the next morning, the defect had already been fixed and all the Lotus Notes scanner tests passed on the next automation run.

4.2 Test Scripting Efforts

Including developers and manual testers in the test scripting process has many benefits including increasing the number of automated tests and improved understanding of automation results.
If everyone on the team has a good understanding of test script structure and framework functions, it is easier to interpret automation results without relying on assistance from automation engineers. Also, manual testers can leverage the framework functions to write scripts that assist them with repetitive testing tasks. Scripts can also be a useful way to reliably reproduce defects. We briefly discuss two efforts to include developers and manual testers in the test scripting process. We plan to continue these efforts in the future and train all developers and QA engineer to write test scripts.

At the Bangalore site, the automation engineers worked with developers to automate a set of tests. Each developer was assigned around 5-10 test cases to automate and had assistance from automation engineers to answer questions, fix bugs, and add new functionality to the test framework as needed. The developers have strong programming skills and learned the framework quickly. However, there was a learning curve to understand test script best practices such as setup, cleanup, and appropriate conditions to verify. All of the scripts were reviewed by automation engineers and after minor fixes most were added to the FVT automation suite.

The Beaverton automation team has also successfully trained two QA team members to use the automation framework. These efforts started when QA engineers completed their iteration tasks early and had time to devote to other tasks. Since there are hundreds of automatable test cases, the automation team asked them to help write test scripts for some of these tests. Beaverton automation engineers assisted QA team members with setting up an automated testing environment and gave them a brief tutorial and script examples. After writing a few test scripts, QA engineers were quickly brought up to speed on the automation process and best practices. The engineers have automated a large number of

tests and continue to actively contribute to automation efforts. Training QA engineers to automate tests has significantly increased the productivity of the automation team and is a good way to leverage the spare cycles of the engineers when they do not have higher priority QA tasks.

5 Conclusions and Future Directions

We have shown the success of preliminary efforts to include all team members in automation and have shown ways that all engineers can contribute to and benefit from automation. The automation and reporting framework has enabled rapid defect detection and 20% of product defects have been detected by automated tests.

Our future plans include improved reporting features including linking failures to known defects and identifying new and unexpected failures. This information will help all team members better understand test results and prioritize troubleshooting efforts. We would also like to link test to specific product features to help team members more quickly identify the root cause of failures. The goal is to identify the components and functionality covered by each test script and store this information in a database. When one or more tests fail unexpectedly, we could query the database to determine which components may have caused the failure. Similarly, if a developer checks in changes to a component, they could determine which tests are affected by the change and monitor the results of those tests in the next automation run. Finally, we plan to continue training developers and manual testers to use the automation framework and hope to continue to increase the productivity of the entire team through automation.

Acknowledgments

Praveen Soraganvi provided valuable feedback on the topic and scope of this paper and on an earlier draft. Everyone on the Endpoint Security Automation team including Keith Albin, Steve Nguyen, Dale Wacker, Jenny Yu, Anand Iyer, Sudhindra Kembhavi, and Amit Patel contributed to the design and implementation of the end-to-end automation and reporting framework described in this paper. We thank everyone on the VSE development and QA teams and the MOVE development and QA teams for their feedback on the automation framework and for successfully incorporating the framework into their projects.