Continuous Model-Based Verification of the Linux Kernel

PROJECT PLAN

Team #9
Client/Adviser: Dr. Kothari
Srinivas Dhanwada - Team Leader
Collin McIntyre - Scribe, Tool Integration Leader
Matt Wall - Web Leader
Ben Weno - Automation Leader
sdmay18-09@iastate.edu
http://sdmay18-09.sd.ece.iastate.edu/

Revised: 10/30/2017 | Version 2

Table of Contents

List of Figures

List of Tables

List of Definitions

MBV - Model Based Verification

kernel - Core of an operating system

1 Introductory Material

1.1 Acknowledgement

Our team would like to acknowledge Dr. Suresh Kothari and his PhD. student assistant Payas Awadhutkar. Both have provided and will continue to provide valuable technical advice and access to various resources that will be crucial to the completion of this project.

1.2 Problem Statement

Iowa State University's Knowledge Centric Software Lab has developed a tool to verify the Linux kernel for specific types of bugs. Unfortunately, in its current state the tool requires a patch each time time the kernel is updated, and must be run manually. Because Linux is used in many applications, it is important to find any bugs that could cause problems as quickly as possible.

Our team will develop a system to automatically create a patch and run the tool every time the the kernel is updated. Also, as a way to make finding bugs quicker, we will create a website that will post the results of each run, and allow users to view and verify results.

1.3 Operating Environment

Our product's operating environment is purely virtual since we will be producing only software. Linux kernel verification will take place on a single machine and the results will be displayed online, but there is currently no plan for a physical product.

1.4 Intended Users and Intended uses

Our intended users include, but are not limited to: The Linux kernel development team, software developers and researchers, and anyone interested in the integrity of the Linux kernel.

Our product only has one intended use: verify the integrity of the Linux kernel. This will be achieved through an automated process where a patch will be generated for the kernel that will allow the verification program to run as intended, the verification program will run and generate data explaining what parts of the Linux kernel are verified safe and unsafe, and this data will be pushed to a server where users can view results and verify their accuracy.

1.5 Assumptions and Limitations TODO

Assumptions

- Users of the website will be willing and able to assist in kernel verification
- The verification program works as intended
- No sweeping changes to the organization of the Linux kernel will be made

Limitations

- The verification process must complete within 24 hours
- Web platform must be widely supported
- The verification program provided must be incorporated into the automation process

1.6 Expected End Product and other Deliverables

1. Automated support for running the MBV toolbox on new versions of Linux. This includes the tool to create patches for the new versions, and the automation of the entire verification process. We will also determine which control paths have changed, and only consider those for viewing.

2. A public website to host the the verification evidence and facilitate collaboration for the verification process.

2 Proposed Approach and Statement of Work

2.1 Functional requirements

- Automatically recognize kernel updates, and start the tool
- Create and apply a patch for the tool for each new version of the kernel
- Run the tool with the new patch, and find changes to the kernel where possible bugs could have been added
- Post the results to the website
- The website must properly display the new results in an organized manner

2.2 Constraints considerations

- The system must operate with a high level of usability for the intended end user
- Parts of the project written in the same language must follow the same coding standard

2.3 Technology considerations

The main piece of technology we will be using during this project is the Linux verification tool. This is based on Atlas, a plugin for Eclipse that can generate models based on code paths. This will lock us into using Java for the project, however we are all familiar with it, so this shouldn't be too much of a problem.

2.4 Safety considerations

As a software project, there are no safety concerns for this project.

2.5 Previous Work and Literature

Our project is based on automating L-SAP, a tool that verified the Linux kernel. We will have some documentation and some results from that tool to use to help us in creating this project. L-SAP verifies the kernel by searching through control paths, and checks to see if 2 part problems such as mutex locking will run both lock and unlock in all paths. So far, the tool has been run 3 times by manually creating patches for each of the 3 versions of the kernel. Other information is available on the L-SAP homepage. The tool runs using ATLAS, an eclipse plugin that enables the exploration of control paths in c code.

2.6 Possible Risks and risk management

Our team is relatively unfamiliar with the existing verification infrastructure (i.e. the inputs and outputs associated with the verification program that is provided to us). This could slow down the development process as we attempt to integrate the program into our automated process.

2.7 Project Proposed Milestones and evaluation criteria

Two possible milestones are the completion of the automation of the system, and the integration of the website with our automated system. We could test the first by observing the results of the system after a new update to the Linux kernel is put out, as well as unit based testing. We can test the website by viewing it's output, and verifying that website navigation works properly.

2.8 Project tracking procedures

We will be using GitLab and its many features to assign tasks, discuss progress, and facilitate source control during the development process.

2.9 Objective of the task

Our objective is to automate the Linux kernel verification tool, and provide a website to check its results easily.

2.10 Task approach

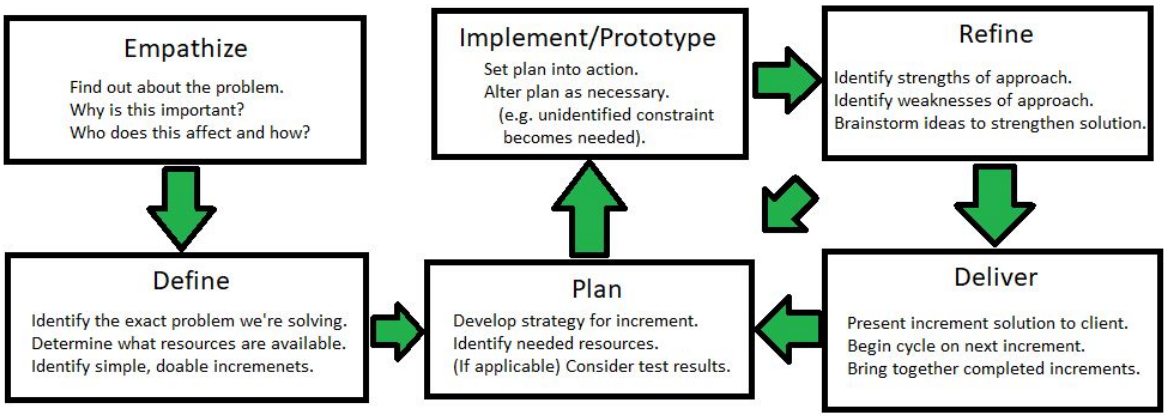

Figure 1

We plan to gain a deeper understanding of why the project is important to the client. From this point, we will identify sub-tasks (labeled "increments" in Figure 1). For each

increment, we will decide which developer will complete a given part of the increment based on skills and interest, create the increment, then identify areas that need refining. Once the increment is complete, we will combine it with previously completed increments. We will follow this process until all increments are complete.

2.11 Expected Results and Validation

We have been given the results of running the verification tool on three versions of the Linux kernel. By running our automated tool/process on those same three versions, we can expect that the automated tool works as intended if our data matches the data generated from the previous runs. For the website, we can be sure that the website works as intended if the data we receive from running the verification tool is the same data displayed on the website.

3 Estimated Resources and Project Timeline

3.1 Personnel effort requirements

Task	Breakdown	Expected Time Commitment
Automatic Patch Generation	This will require a deep dive on how patches are generated, and then research to find a system to automatically create them. Finally, the system must be put into place and tested.	4 weeks
Automate existing tool	This will require research in how the current tool works, and implementation of automating it. We will also need to find and implement a way to kick off a new run of the tool when a new kernel is released. Finally, we will need to create tests.	5 weeks
Filter models that have been updated since the last run of the tool	We must create a system that will filter areas where the kernel was changed, so that we don't have to check every part of the kernel at every update. This will require research on how the tool currently outputs, and then implementation of the system and testing.	3 weeks
Post results to website	We will need to find a way to get the results to the website, and package them in a meaningful way.	2 weeks
Display results on website	This will be creating a website that is functional, and easy to navigate to host the results from the tool.	8 weeks
Documentation	Creating documentation for the automation of the tool.	2 weeks

Table 1

3.2 Other resource requirements

We will need access to patches that were applied to previous versions of the Linux kernel and access to the current website that is used by the Knowledge Centric Software Lab. These resources are provided by the client and will allow us to develop a solution that integrates seamlessly with the infrastructure that is already in place.

3.3 Financial requirements

All resources are provided by our client. As such, there are no additional financial requirements. The only true cost associated with this project is the cost of upkeep of the machine that will run the automated process, and the upkeep of the server that will display the webpage.

3.4 Project Timeline

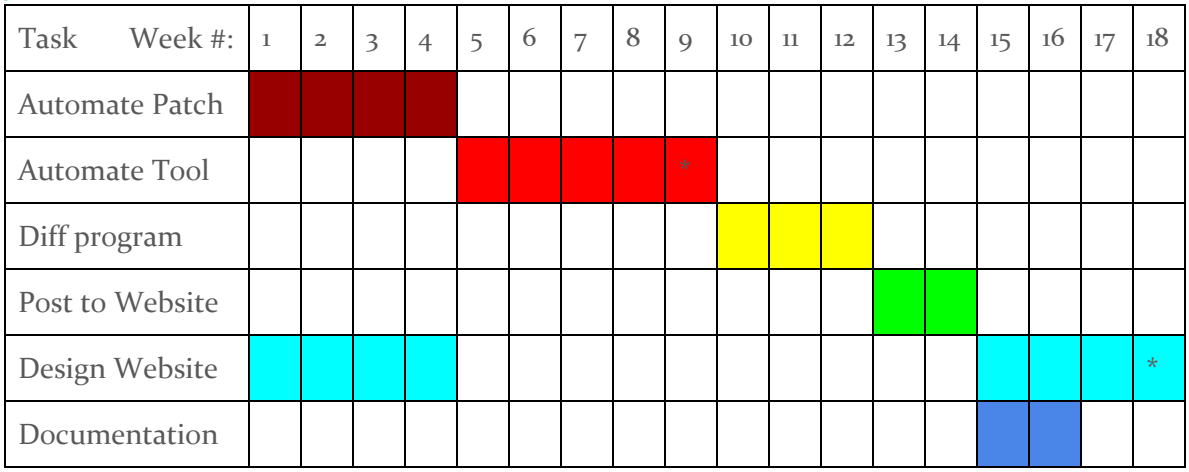

Figure 2

For both of our deliverables, our Gantt chart above (Figure 2) shows that at the end of week 9 we will have finished with automating the tool. Then, at the end of week 18 we will have finished the website.

4 Closure Materials

4.1 Conclusion

Making sure the Linux kernel is bug free is important because of the sheer number of applications that use it. We plan to aid the process of detecting bugs by automating this tool, and posting the results to a website, where bugs can be reported, and eventually fixed. We hope that our work will provide a more bug-free world.

4.2 References

Project proposal document - http://bit.ly/2hoSuRO

L-SAP tool - http://home.engineering.iastate.edu/~atamrawi/l-sap/index.html

ATLAS tool - http://www.ensoftcorp.com/atlas/