Survivability of Operating Systems: Handling Vulnerabilities

Neeraj Suri and Andréas Johansson
Department of Computer Science, TU Darmstadt, Germany
Email: {suri, aja}@informatik.tu-darmstadt.de

Dependability and security are flip sides of the same coin – linked conceptually though differing in process! Both arise from the occurrence of design or operational "deficiencies" in a system though differ in the processes of invocation of these deficiencies. Security issues often pertain to the intentional abuse of these deficiencies (vulnerabilities) to perturb the system operations; dependability, on the other hand, leans more towards handling of the unexpected and unintentional exercising of such deficiencies (faults & errors) – the consequence of vulnerabilities from either the security or dependability viewpoint being that the delivery of expected services gets (potentially) disrupted. In this sense, the term "vulnerability" is equally applicable from both the dependability and security perspectives; we will use this term interchangeably representing fault/errors in the dependability context, and also its more common security usage. For the scope of this write-up, our "survivability" viewpoint focuses on enhancing the robustness of systems to provide for sustained delivery of proper services, be they dependability or security nuanced.

Given the fundamental role an operating system (OS) plays in any computing system, we target survivability issues at the OS and related middleware levels. Our current research targets ascertaining specific design/operational vulnerabilities, and equally importantly two related aspects that constitute the focus of this write up, namely (a) ascertaining the propagation profiles of data errors across the system assuming the existence of vulnerabilities, and (b) conducting error impact or consequence analysis given the (real or speculated) presence of a vulnerability. Studying the impact of speculated or real vulnerabilities allows us the possibility to forecast the *effect* they might have on the provided services. This impact analysis can be used, via backtracking, to locate possible vulnerabilities that would lead to consequences of graded severity, and to predict the impact of vulnerabilities exposed after the system is deployed.

Although identification of specific vulnerabilities is a key ingredient for survivability planning, we maintain that generation of error propagation profiles aids us better understand the actual impact and damage an exercised (intentional or unintentional) vulnerability can have. Consequently, one can conduct focused composition and location of effective wrappers to mitigate the damage. Of course, if a specific vulnerability gets identified, or its likelihood of occurrence is known (corresponding to the data error propagation profile in a system), the more effective the use of wrappers is in helping enhance the system survivability.

In our ongoing work, we focus on these issues for static, modular embedded software and created a framework for profiling error propagation and errors effect to aid effective wrapper placement. The framework is called ***EPIC*** after the measures it introduces of ***E****xposure* of a SW component to propagation of errors, ***P****ermeability* of a SW component to incoming errors, ***I****mpact*, and ***C****riticality* of an error percolating through the system and affecting requisite functionality. The EPIC framework, supported by a tool environment ***PROPANE*** (***PROP****agation* ***AN****alysis* ***E****nvironment*), enables the estimation and quantification of error propagation and error effect profiles. Propagation profiles reveal information on where errors propagate through the system and which modules/signals are more exposed to propagating errors. These profiles use the basic measures of permeability and exposure for modules and signals. To cover the cases where an error is unlikely to occur, but potentially has a high impact on system services if it does; the error effect profiles consider the effect/impact errors have on the system outputs. The impact profiles provide this information. Weighing the output signals according to their importance aids accounting for different output criticalities as well. The EPIC framework was evaluated, using fault injection experiments, on a real embedded software system. The aforementioned measures were estimated and used in the placement of error detection/handling mechanisms, i.e., wrappers.

OS Level Profiling and Wrapping: Issues, Approaches, and Opinions

For static embedded software, a proscribed set of SW modules is usually known along with their predefined interactions; this makes it possible to find all communication paths in the system and then profile accordingly. At the level of OS's, middleware and applications, the situation is considerably harder. As the set of applications is not generally stipulated, consequently enumerating all possible interaction paths (and consequently error propagation paths) is hardly viable. Thus, a dynamic model must be used where different sets of applications can be used without needing to redo the profiling for each change. At the same time, the OS (and application SW) is composed of components, with well defined interfaces where the propagation of errors can be studied. Within the OS setting, our focus is on establishing how errors propagate from drivers to application level services. Drivers are a major part of modern OS's and unfortunately also considered to be one of the largest sources of errors in an OS's, and warranting focused attention. Thus, we intend to characterize the interactions taken between the OS and its drivers, focusing on how errors in drivers affect, i.e., propagates through, the OS. We will intend to compare the impact of propagating errors by classifying the result of propagations, such as data level propagation, process failure and complete OS failure.

We aim to use the profiling information for composing wrappers to elevate the service level robustness. In using wrappers, the necessary prerequisites for developing and providing the relevant level of robustness/dependability in software involve (a) the type of errors the system is supposed to handle; their nature, frequency, duration, etc. (b) the available mechanisms for dependability – specifically failure detectors - and their properties, and (c) the characteristics of software with regard to vulnerabilities and hot-spots to effectively incorporate available dependability mechanisms where they are most effective, and where errors propagates and do most damage. It is important to state that the accuracy and representativeness of the error model fundamentally determines the consequent accuracy of any profiling activity.

We believe that quantitative approaches are called for to solve this problem, due to both the sheer size and complexity of modern OS's as well as their inherent dynamic nature. As the overall goal of any computing device is to provide services to its users, the final goal of our profiling is to estimate the impacts errors have on the services provided by applications. The problem is that profiling with respect to a certain set of applications running on the system will not give the same result as a different set of applications. The way applications use the OS and their importance will influence the results of the profiling. To capture this distinction we propose to use two separate profiles, an OS profile and a separate profile for an application's use of the OS. This profile must include information revealing which OS services the application depends upon and some notion of each service importance to the application (some are most likely more important for the continued functioning of the application than others). To analyze a system with more than one application, with varying importance, priorities must be established across them. This can then be used when the assignment of wrappers is made.

The proposed OS profiles consider how errors in drivers spread through the OS to the services they provide for applications. They do not however, consider how those service errors affect applications using them. This means that the OS profiling measures are specific with respect to an OS and its drivers. These measures can be used to find out which OS services are more susceptible to propagating errors or which drivers are more likely to spread them. Three measures will be defined that collectively constitutes the OS profiles. First the permeability of the OS is measured, i.e., quantifying the ease of error propagation through the OS. Then this measure is used to define the subsequent measures, detailing the exposure of errors on OS services and diffusion of errors from drivers. When adding wrappers, the more likely or more severe propagation paths will be considered. However, a wrapper is only considered if it protects against propagating errors relevant to the applications running on the system. By profiling applications interactions with the OS and composing this information with the error propagation profile we will extract only the relevant wrappers needed. As every system has a limited set of resources, in performance and size, a trade-off is called for where these properties play a major role. By profiling the OS, and then on the basis of the applications running, tailoring the relevant components of the system we will get a system that both has a higher degree of robustness and flexibility.

The suggested profiling approach is currently based on experimental evaluation techniques. We intend to introduce errors at the interface level between drivers and the OS and study their effects on the application level services. Like previous error propagation studies, e.g., ***EPIC***, we intend to use fault injection as method for evaluating our measures. As with any experimental method it introduces coverage limitations. Our method is naturally limited to the error model chosen. Any error propagation profiling is always made with respect to an error model and it is with this in mind that one should interpret the profiling results. Another issue to keep in mind is that there exist dependencies between the application and the OS that we currently do no cover. For instance we do not consider the order in which calls are made by an application to the OS. The ordering of calls, with respect to errors, might have an impact on the resulting behavior of the application. Investigating such dependencies and their implications for wrapper placement is part of future extensions for our work.

For determining if an error propagated or not, one needs to know what constitutes an error for the delivery semantics of the services. This information can (ideally) be derived from the service specifications. This is not always possible due to the lack of or incompleteness of given specifications. In the past many fault injection experiments have utilized a so called golden run, i.e., a test program is executed without faults and the references outcome is compared with the run with faults presents. Any deviation from the golden run is considered as erroneous. For OS's, creating a golden run is difficult due to non-determinism in scheduling etc. One option would be to restart the system before every experiment and then conduct the tests (golden runs as well as fault injection experiments). Another option would be to run several tests without faults and use a mean behavior as the golden run.

Regardless of the specific approaches used to ascertain vulnerabilities or profiling their effects, the issues of survivability constitute a multi-dimensioned problem that needs to be addressed over multiple abstractions levels of a system. Our take on the problem is, at the OS level, to establish error propagation profiles (along with quantitative measures for the profiles) as an aid to establishing the effect vulnerabilities can have on system services, and using selective use of wrappers to mitigate the potential for disruption of desired services, i.e., enhance system survivability. It is the handling of the consequences, as vulnerabilities get exercised, that motivates our approach to survivability. A few general opinions:

- We need to start addressing issues of vulnerabilities, survivability etc at SW architectural levels transcending from the current thrusts targeted more at the program/code levels.
- Does access to source code provide any meaningful benefits versus working with operational/behavioral specifications of systems? Given the complexity of systems, perhaps we should focus more on grey-box level techniques. Similarly, perfection of survivability attributes at a sub-system level does not necessarily translate to system level survivability – are there any survivability compositions that need to be developed for scalability of survivability?
- Quantification and measures for vulnerabilities, service impairments, and security are a definite need, and the current measures in certain need of refinement/enhancement!!
- Security and Dependability offer an interesting interplay for survivability issues; focusing on effects of vulnerabilities might aid development of synergies across these two seemingly disparate fields!