A Near-miss Management System to Facilitate Forensic Investigation of Software Failures

M.A. Bihina Bella, J.H.P Eloff, M.S. Olivier
University of Pretoria, Pretoria, South Africa
mbihina@yahoo.fr
jan.eloff@sap.com
martin@mo.co.za

Abstract: The increasing complexity of software applications can lead to operational failures with disastrous consequences. In order to prevent the recurrence of such failures, a thorough post-mortem investigation is required to identify their root causes. The root-cause analysis must be based on reliable digital evidence to ensure its objectivity and accuracy. However, current approaches to failure analysis do not promote the collection of digital evidence for causal analysis. A promising alternative is the field of digital forensics. Digital forensics uses proven scientific methods and principles of law to determine the cause of an event based on forensically sound evidence. However, as a reactive process, digital forensics can only be applied after the occurrence of costly failures. This limits its effectiveness as volatile data that could serve as potential evidence may be destroyed or corrupted after a system crash. A more proactive approach to digital forensics is therefore required. The analysis of near misses is a promising solution to the above issue. Unlike failures, near misses do not result in loss. Instead, they are high risk situations with potential for loss or damage, and as such, are often forerunners to serious failures. The detection of near misses therefore provides an opportunity to safely collect relevant failure-related data before the failure occurs. Near-miss analysis has been implemented successfully for decades in many engineering disciplines, but it is not yet used in the IT industry. The current paper therefore proposes the architecture of a near-miss management system suitable for the software industry. The paper first proposes a definition of a near miss from an IT system perspective. The proposed definition is based on the allowed downtime indicated in the Service Level Agreement (SLA), which specifies the system's contractually agreed performance level. The downtime-based definition of near misses is then used to detect and classify near misses based on their risk level.
.

Keywords: failure, near miss, digital forensics, SLA, downtime

1. Introduction

Software failures have occurred since the beginning of computers as evidenced by the number of highly publicised IT accidents reported in the media. One recent example of a crisis caused by a software bug is the system failure at RBS (Royal Bank of Scotland), a major bank in the UK, in December 2013. Due to an unspecified technical glitch, the bank's various electronic channels were unavailable and customers were unable to make payment or to withdraw cash with their debit cards (Finnegan, 2013). This outage occurred on "Cyber Monday", which is considered one of the busiest online shopping days of the year. This failure occurs after a major outage in June 2012, which left millions of customers unable to access their bank accounts for four days due to a failure in a piece of batch scheduling software. As a result, deposits were not reflected in bank accounts, payrolls were delayed, credit ratings downgraded and utility bills not paid (Worstall, 2012).

The examples quoted above show that software failures result in downtime and poor system performance which can have disastrous consequences. Preventing the recurrence of such disasters is crucial. This requires finding out their root cause, so that the associated errors and constraints in the design or configuration of the software can be corrected. To this effect, a thorough post-mortem investigation must be conducted. In order to ensure the validity of its results, the investigation must be based on reliable digital evidence such as log files, system settings and database dumps.

Sound evidence of the software failure promotes the objectivity and comprehensiveness of the investigation, which translates into a greater accuracy of the results. Furthermore, reliable evidence is also valuable in the event that the software failure leads to a product liability lawsuit. However, current informal approaches to failure analysis do not promote the collection and preservation of digital evidence. Rather than depending on objective evidence analysis, failure analysis methods often rely on the investigator's experience with the system to identify the cause of the problem. Examples of such failure analysis approaches include application performance management tools, transaction management tools, the war room method as well as troubleshooting (Neebula.com, 2012).

Troubleshooting in particular, which is usually the first response to a system failure, focusses on restoring the system to its operational state as quickly as possible. This allows little time and resources to collect evidence of the failure. Besides, system restoration often requires rebooting, which destroys or tampers with valuable information that could pinpoint the root cause of the problem (Trigg & Doulis, 2008). Both these issues leave the system vulnerable to the recurrence of a similar failure.

The 2012 outage at RBS mentioned earlier illustrates the above point. Indeed, initially the origin of the problem was identified through troubleshooting but the corresponding patch which was applied some hours later did not lead to a full system recovery (Du Preez, 2012). This suggests that the troubleshooting approach did not identify the root cause of the problem. The real source of the glitch was discovered after a comprehensive investigation from an independent expert was ordered by the Financial Services Authority (Finnegan, 2013).

In order to ensure that the failure analysis is based on reliable evidence, the investigation must follow a process that favours the collection and analysis of such evidence. The forensic process is well suited for this purpose. Indeed, forensic science uses objective evidence to reconstruct past events. The field of digital forensics, as the application of forensic science to digital systems, also follows established procedures meticulously to ensure the accuracy and completeness of digital evidence and to interpret it objectively (Vacca and Rudolph, 2011). We argue that it can therefore serve as an effective alternative to investigate major software failures, although it is currently limited to the investigation of criminal events and security incidents. Various standards such as the ISO/IEC 27002 information security standard (ISO/IEC, 2007) and the American National Institute of Standards and Technology (NIST) (Scarfone et al. 2008) support this view and recommend the collection of forensic evidence for the investigation of system failures.

However, as beneficial as it is, digital forensics follows a reactive process which can only be applied after the occurrence of a costly failure. This limits its effectiveness as volatile data that could serve as potential evidence may be lost after a system crash. In order to address this limitation of digital forensics, we propose to start the forensic investigation at an earlier stage, before the software failure completely unfolds. This requires the detection of high risk conditions that can lead up to a major failure. Such forerunners to failures are known as near misses in the risk analysis literature (Jones et al. 1999).

By definition, a near miss is a hazardous situation where the sequence of events could have caused an accident if it had not been interrupted (Jones et al. 1999). This interruption can be caused by chance or human intervention. A simple example of a near miss in everyday life is the case of a driver crossing a red traffic light at a high speed at a busy intersection without a collision. As a near miss is very close to an entire accident sequence, it provides a fairly complete set of data about the ensuing accident. This data can be used as evidence to reconstruct the accident and to identify its root cause. The analysis of near misses can therefore offer a proactive way to safely collect relevant failure-related data before the failure occurs. For this reason, we explore the application of near-miss analysis to facilitate the forensic investigation of software failures.

Although not yet used in the software industry, near-miss analysis is an established field successfully used in several engineering fields and in the military and healthcare industries. It is usually performed through so-called near-miss management systems, which are automated tools designed to record and analyse near misses (Phimister et al. 2003). The current paper therefore presents our work-in-progress for the architecture of a near-miss management system designed for the software industry.

The proposed architecture follows the digital forensic process to collect evidence of the event and determine its root-cause. The architecture also accommodates design challenges specific to the software industry where the concept of a near miss is still largely unexplored. The architecture therefore lies at the intersection of failure analysis, digital forensics and near-miss analysis as illustrated in Figure 1.

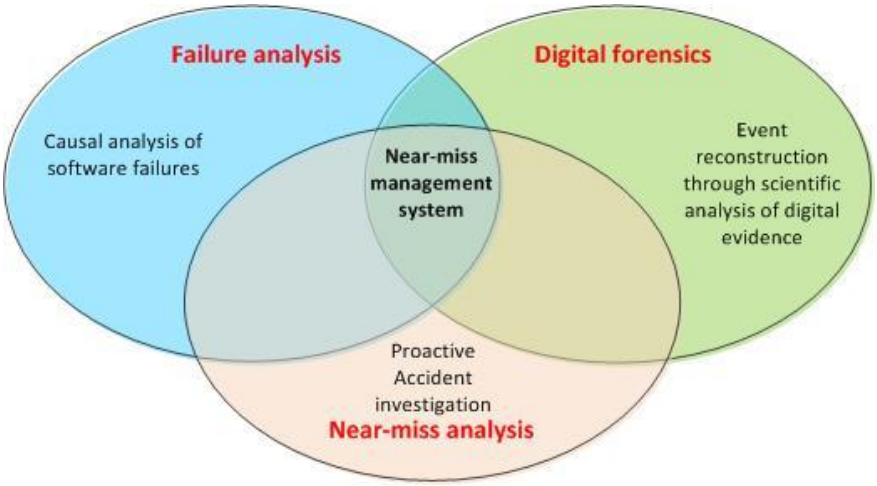

Figure 1: The proposed near-miss management system in relation to the fields of failure analysis, digital forensics and near-miss analysis

The remainder of the paper is organised as follows. Section 2 discusses challenges to near-miss management. Section 3 reviews previous work on near-miss management to address these challenges and assesses its suitability for IT systems. Section 4 presents our proposed solutions to the above challenges from an IT system perspective. Finally an architectural representation of the overall near-miss management system designed to implement the proposed solutions is presented in Section 5.

2. Challenges to near-miss management

Across industries, the successful implementation of a near-miss management system faces two main challenges: (1) the detection of events and conditions that can be classified as near misses and (2) the high volume of near misses. A discussion of these challenges follows.

1) Detection of near misses: What exactly constitutes a near-miss event varies between industries and organisations. Therefore organisations need to clearly define and specify which type of hazardous events should be reported as near misses. This is particularly challenging in the software industry. Indeed, in other industries, near misses are often obtained from observed physical events and conditions. In the case of software applications, some near misses might not be visible as no system failure occurs.
2) High volume of near misses: Near misses are typically more frequent than failures and can be numerous. In fact, an extensive study of industrial accidents conducted in 1969 indicates that a severe injury can have up to 600 near misses as precursors (Nichol, 2012). This is shown in the popular accident ratio triangle in Figure 2. A more recent study conducted in 2003 suggests that this number could be even higher (Nichol, 2012). A high volume of near misses is also expected in the software industry, as reports of various major software failures indicate that these disasters are often preceded by several minor events. With limited investigative resources, the sheer volume of near misses can become unmanageable. An effective data reduction mechanism is therefore required to investigate only near misses that offer the most complete digital evidence of the potential failure.

Figure 2: Bird's accident ratio triangle, adapted from Nichol (2012)

Finding solutions to these challenges has been the focus of previous work on near-miss management. This is reviewed in the next section.

3. Previous work on near-miss management

Typically, a near-miss management system performs the following three consecutive main processes:

- Identification of near misses
- Selection and prioritization of near misses for analysis
- Causal analysis of the selected near misses

Previous work on near-miss management therefore focusses on these three areas. As mentioned earlier, the first two are particularly challenging, especially for software systems. However, previous work on near-miss identification is very industry-specific and is not applicable to the software industry. We therefore review previous work on the last two processes. This review is conducted in order to identify previous solutions that could be used in the design of the near-miss management system we propose in this paper.

3.1 Previous work on near-miss prioritisation

Various approaches are used to classify and prioritise near misses. Besides manual screening, risk-based classification and mathematical modelling can be used.

Risk-based classification ranks near misses based on the severity level of their potential consequences or on their frequency (Greenwell *et al*, 2003). This can be performed with a risk decision matrix (Ritwik, 2002) or historical data of reported near misses to determine trends in the occurrence of certain near-miss events (Phimister et al. 2004).

Mathematical modelling can be used to estimate the conditional probability of an accident given a near miss in order to assess its level of severity. This accident probability is defined as the level of closeness between the near miss and the likely accident. Probabilistic risk analysis (PRA) and Delphi techniques can also be used for the same purpose. PRA consists of estimating the risk of failure of a complex system by breaking it down into its various components and determining potential failure sequences. The Delphi method is a group decision-making tool that can be used to obtain information on the probability of an accident from a panel of experts (Phimister et al. 2004).

Although the above techniques can provide useful results, previous work in near-miss prioritisation is generally specific to the industry concerned and often requires prior knowledge about near misses from historical data. The latter is not available in the software industry, where the concept of near miss is still new. However, one valuable concept that can be applied in the IT industry is the prioritisation of near misses based on the likelihood of a failure. Our suggestion on how to use this approach is presented in Section 4.

3.2 Previous work on near-miss causal analysis

Causal analysis of near misses can be performed with investigation techniques from engineering disciplines such as fishbone diagrams, event and causal factor diagrams, event tree analysis, fault tree analysis and failure mode and effects analysis (Reality Charting, 2013). The investigation consists of answering a series of questions that give insight into the factors that led to the near miss, the near-miss possible adverse consequences and the factors that prevented or limited those consequences. The investigation can be assisted by various tools such as a comparative timeline to organise data and various matrices such as the missed-opportunity matrix and the barrier-analysis matrix (Corcoran, 2004).

Statistical analysis has also been proposed for learning from near misses. Some examples are using estimation techniques, simulations and regression analysis (Mürmann and Oktem, 2002). However, as valuable as these techniques are, they do not follow forensic principles. Thus they are not suitable for our proposed near-miss management system. This system applies the digital forensic methodology to analyse near misses, the same way we suggest using digital forensics to investigate software failures. Our proposed process for this purpose was described in a previous paper (Bihina Bella et al. 2011).

As the above review shows, some work is still required to define, classify and prioritise near misses from an IT system perspective. Our vision on how this can be done is explained in the next section.

4. Definition and prioritisation of near misses for IT system

4.1 Definition of near miss for IT system

The exact definition a near miss depends on the industry and organization in hand. However, when referring to engineering products, a common understanding is that near misses are risky conditions or events that can lead to a costly failure if not remedied (Jones et al. 1999).

With regard to IT systems, a failure is "the inability of a system or component to perform its required functions within specified performance requirements" (IEEE, 1990). Since no system is immune from a malfunction, no system vendor can guarantee that the system will work perfectly and continuously at all time. In other words, some periods of downtime are expected. For this reason, the performance requirements of a typical IT system make provision for a downtime "allowance". This allowed downtime can be indicated informally in the system's specifications document but it is usually formally specified in a contract between the service provider and the receiver of the service (customer). This contract is referred to as a service level agreement (SLA) (Sevcik, 2008).

For instance, the SLA for a web site may specify that the site will be operational and available to the customer at least 99.9% of the time in any calendar month. This indicates that the website should not be down for more than 0.1% of the time in a month. For a 30-day month, this corresponds to a downtime limit of 0.03 day or 43 min and 12 s. If the website is down for more than this amount of time in a month, it does not meet the customer's expectation in terms of the SLA. It violates the SLA and is considered to have failed.

Therefore, for the purposes of the research in hand an event is considered a failure if its resulting downtime exceeds the downtime allowance specified in the SLA. Similarly, an event is considered a near miss if it can lead to exceeding that allowance. We therefore propose the following informal definition of a near-miss:

A near miss isan unsafe event or condition that causes a downtime whose duration is close to exceeding the downtime allowance specified in the SLA.

We propose to use the above concept as the basis to formally define a near miss. In order to do so, we need to determine how close the experienced downtime should be from exceeding the allowed downtime to be considered a near miss. We suggest specifying a near-miss threshold for this purpose. This threshold will vary from one organization to the next, depending on its risk tolerance. For instance, a 95% threshold (95% of the downtime allowed) would correspond to a total monthly downtime of 41 min and 2 s. In this case, a near miss is any monthly downtime of between 41 min and 2 s (the threshold) and 43 min and 12 s (the allowed downtime). This threshold-based definition of a near miss can be mathematically expressed as follows.

$D_{experienced}$ is the experienced downtime
$D_{allowed}$ is the SLA downtime allowance
α is the near-miss threshold in percentage; $\alpha < 1$
$\alpha \times D_{allowed}$ is the near-miss threshold in time value
If $\alpha \times D_{allowed} \leq D_{experienced} \leq D_{allowed}$ then $D_{experienced}$ is a near miss

Figure 3 shows the downtime-based classification of events explained above.

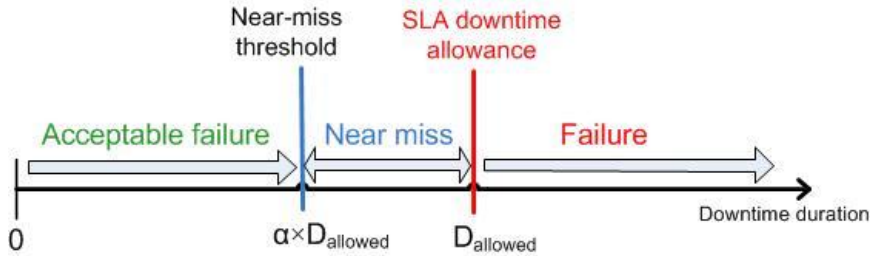

Figure 3: Classification of unsafe events based on their downtime duration

The above formal definition of a near miss enables the detection of near misses among periods of downtime. Once near misses have been identified, they need to be classified and prioritised for data collection and analysis. The next section explores our suggestion on how to prioritise near misses.

4.2 Near-miss prioritisation

We previously defined a near miss as a potential failure, more specifically as an event that can lead to the violation of the SLA. We also defined the violation of the SLA in terms of the system downtime. According to the same logic that we used to measure the severity of a failure based on the downtime experienced, we can assess the severity of a potential failure or near miss based on its expected downtime. In other words, we want to determine for how long the system will be down in the eventuality of an outage caused by this near miss.

To this effect, we need the failure probability of the near miss and the expected recovery time for the outage or MTTR (mean time to repair). The expected downtime is then calculated as the product of the failure probability and the MTTR. The MTTR can be obtained from the system vendor specifications or through historical observations. We are currently exploring formulas from the reliability theory of IT systems (Holenstein, 2003) to develop a suitable failure probability formula.

The system enters a "critical zone" when the expected downtime is greater than the SLA downtime allowance. This can be expressed as the following formula:
$D_{expected}$ is the expected downtime due to a failure
$D_{allowed}$ is the SLA downtime allowance
P is the probability of failure given the current unsafe situation
MTTR is the expected recovery time following an outage
$D_{expected} = P \times MTTR$
If $D_{expected} > D_{allowed} \rightarrow$ critical zone

Only events in the critical zone are passed on for analysis. These events are prioritised based on the value of their expected downtime. If we manage to prevent a system outage once the system is in the critical zone, then this event is classified as a near miss, otherwise it is a failure. Both cases need to be investigated to identify their root cause and prevent their reoccurrence. In the case of the failure, the root cause analysis is facilitated by the fact that the event data collection occurred before the system failed.

The near miss management system that integrates the above steps is presented in the next section.

5. Architectural representation of near-miss management system

This section describes the architecture proposed for a near-miss management system that implements the steps for detecting and prioritising near misses, as well as the data collection and causal analysis described earlier. It starts with an overview of the overall near-miss management process. This is followed by a detailed description of the architecture designed to automate this process.

5.1 The near-miss management process

To detect near misses, the system needs to be monitored with a view to recording and reviewing event logs. It is suggested that events be logged in a central repository such as a Syslog server. By default, Syslog messages are assigned a facility code and a severity value (Gerhards, 2009). The facility code indicates the source of the message (e.g. printer, network) and the severity value indicates its criticality (e.g. warning, error, emergency). This information can be used for a primary categorisation and ranking of unsafe situations. Only severe events and messages that indicate a status change (e.g. up, down, restart) from critical resources are labelled as potential near misses. Events flagged as critical are then sent to another module for prioritisation. This module calculates the system failure probability and expected downtime based on each event.

Afterwards, events identified as being in the critical zone are passed on to another component for data collection. The Simple Network Management Protocol (SNMP) is proposed for this purpose. This Internet-standard protocol enables information exchange between a manager (central unit) and its agents (the other system units) (Presuhn, 2002). In this case, the SNMP manager is the data collection module and it requests additional information about the event from the relevant units of the monitored system through their SNMP agents. Corrective steps are subsequently taken to prevent a system failure, if possible. Finally the collected data is used for causal analysis of the event. Upon identification of the root cause of the event, recommendations are made to correct the system flaw.

The architecture of a near-miss management system proposed to implement the above process is described in the next section.

5.2 The near-miss management system architecture

The proposed near-miss management system architecture is shown as a UML component diagram in Figure 4. The architecture consists of five main components listed below in their logical sequence:

- The Near-Miss Monitor
- The Near-Miss Classifier
- The Near-Miss Data Collector
- The Failure Prevention
- The Event Investigation

Some components are made of several sub-components that all have a type: a document file, an executable file or a database table. Dashed arrows indicate a component's dependency from the source component to the target component. For instance, the Near-Miss Classifier requires high-risk event logs from the Near-Miss Monitor. Some dependencies are subject to conditions, for example an expected downtime must occur in the critical zone to activate a data request from the Near-Miss Data Collector.

Each of the main components processes the event logs from the monitored system. The five main components of the system are used to perform a multi-stage filtering process that progressively discards "irrelevant" events and only retains near misses with the highest risk factor. The key component of the architecture is the module used to prioritise near misses. This module is named the Near-Miss Classifier. It uses the equation established in Section 4.2 to classify near misses based on their expected downtime. A description of the main components of the architecture follows.

5.2.1 The Near-Miss Monitor

The Near-Miss Monitor monitors the units of the system to identify potential near misses. This module is implemented as a Syslog server, which receives logs from every unit. Using predefined criteria (message content, source and risk level), the Syslsog server classifies the logs based on their facility codes and severity values. Events classified as potential near misses are then sent to the Near-Miss Classifier.

5.2.2 The Near-Miss Classifier

The Near-Miss Classifier calculates the failure probability and expected downtime and prioritises events accordingly. Logs of events identified as being in the "critical zone" are sent to the Near-Miss Data Collector and an alarm is raised to notify the system administrator.

5.2.3 The Near-Miss Data Collector

This module is implemented as an SNMP Manager. The SNMP Manager requests data from the units in the critical zone. Such data may include the source identifier (e.g. IP address), running processes, system settings and error messages. This data is then stored in the Event Data table and transferred to the Failure Prevention module.

5.2.4 The Failure Prevention

With this module, the system administrator uses the collected data to identify and implement appropriate corrective steps in an attempt to prevent - or at least mitigate the impact of - system failure. This might include ending some active but unused processes or deleting some stored but unnecessary data to free up memory. The administrator records the steps implemented in a log file for future reference. He then sends the outcome of the recovery attempt (successful or unsuccessful) to the Event Investigation module.

5.2.5 The Event Investigation

Based on the outcome of the failure prevention process, the Event Investigation module classifies events as either near misses or failures and stores the event details in the appropriate table for future reference. The administrator then conducts a root cause analysis of the event based on the data stored. Afterwards, recommendations for improvement are made and implemented either immediately or at a later scheduled

time. These recommendations are stored along with the event details in the relevant table. These steps allow for the creation of an event history that can be looked up in the event of a similar event occurring in the future.

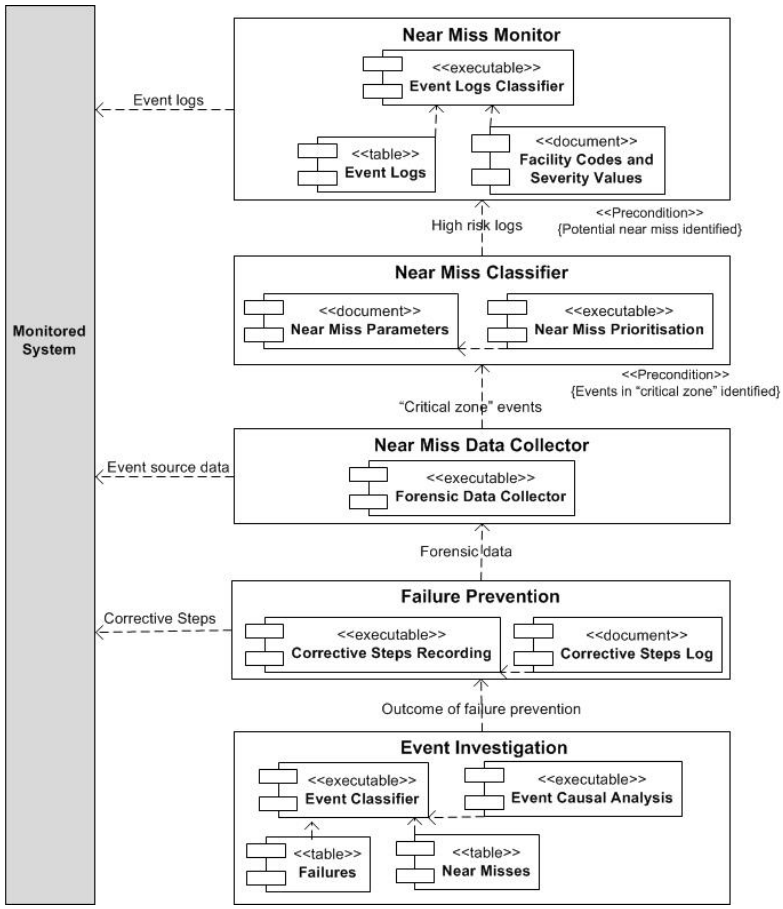

Figure 4: UML component diagram of near-miss management system

This architecture enables the safe and proactive collection of data related to the potential failure for root-cause analysis. Besides, it enables the improvement of the software correctness once the cause of the outage has been established and recommendations for improvement have been implemented. This prevents the recurrence of a similar failure in the future. An additional benefit of the architecture is that it enables the prevention of a failure if appropriate corrective actions are executed timely.

6. In conclusion

This paper examines the application of a near-miss management system to facilitate the forensic investigation of severe software failures. The paper established a definition of near misses for IT systems, and developed a method to detect, classify and prioritize near misses so as to address the issue caused by their high volume. The automation of this process was presented through the architecture of a near-miss management system. This process is work-in-progress and research is underway to refine it. Future work is the development of a mathematical formula to calculate the likelihood of a failure due to a near miss in order to prioritise high risk near misses for detailed causal analysis.

References

Bihina Bella, M.A., Olivier, M.S. and Eloff J.H.P (2011). *Proposing a Digital Operational Forensic Investigation Process*, WDFIA, London, UK

Corcoran. W.R. (2004) "Defining and Analyzing Precursors", *Accident Precursor Analysis And Management: Reducing Technological Risk Through Diligence*, Phimister, J.R., Bier V. and Kunreuther, H., Washington D.C, USA: The National Academies Press, pp 79-84

Du Preez, D. (2012) "RBS failure to fix IT systems after month is 'unprecedented' say analysts" [online],

Computerworld UK, http://www.computerworlduk.com/in-depth/infrastructure/3369901/rbs-failure-to-fix-it-systems-after-month-is-unprecedented-say-analysts/?intcmp=rel_articles;infrstrctr;link_4
Finnegan, M. (2013) "RBS apologises as customers hit by another IT outage", [online], Computerworld UK, http://www.computerworlduk.com/news/it-business/3491865/rbs-apologises-as-customers-hit-by-another-it-outage/
Gerhards, R. (2009) "The Syslog Protocol", RFC 5424, [online], http://tools.ietf.org/html/rfc5424
Greenwell, W.S. Knight, J.C. and E.A. Strunk. (2003) "Risk-Based Classification of Incidents" *In Workshop on the Investigation and Reporting of Incidents and Accidents*, Department of Computer Science, University of Virginia, [Online], http://shemesh.larc.nasa.gov/iria03/p03-greenwell.pdf
Holenstein, B., Highleyman, B. and Holenstein, P.J. (2003) *Breaking the Availability Barrier: Survivable Systems for Enterprise Computing*, Vol. 1, Bloomington, USA: Authorhouse, pp 27-28
IEEE. (1990) *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*.
ISO/IEC. (2007) "Information technology - Security techniques - Code of practice for information security management", International Standard ISO/IEC 27002, Switzerland, Geneva
Jones, S., C. Kirchsteiger, and W. Bjerke. (1999) "The Importance of Near Miss Reporting to Further Improve Safety Performance" *Journal of Loss Prevention in the Process Industries*, Vol. 12, pp.59-67
Neebula.com. (2012) "Success Factors for Root-Cause Analysis" [online], e-Book, http://www.neebula.com
Nichol, K. (2012) "The Safety Triangle Explained" [online], http://crsp-safety101.blogspot.com/2012/07/the-safety-triangle-explained.html
Phimister, J. R., Bier, V. M., Kunreuther, H. C. (2004) *Accident Precursor Analysis and Management: Reducing Technological Risk Through Diligence*, National Academies Press, [online] http://www.nap.edu/catalog/11061.html
Phimister, J.R., Oktem, U., Kleindorfer, P.R. and Kunreuther, H. (2003) "Near-miss Incident Management in the Chemical Process Industry", *Risk Analysis*, Vol. 23, No. 3, pp 445-459.
Presuhn, R. (2002) "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)" [online], RFC 3418, http://tools.ietf.org/html/rfc3418
RealityCharting.com. (2013) "Various RCA Methods and Tools in Use Today" [online], http://www.realitycharting.com/methodology/conventional-wisdom/rca-methods-compared
Ritwik, U. (202) "Risk-based approach to near miss". *Hydrocarbon processing*, pp 93-96
Scarfone, K., Grance, T. and Masone, K. (2008), "Computer Security Incident Handling Guide", *NIST Special Publication 800-61*, Revision 1, [online], National Institute of Standards and Technology, Gaithersburg, USA, http://csrc.nist.gov/publications/nistpubs/800-61-rev1/SP800-61rev1.pdf
Sevcik, P. (2008) "Service Level Agreements for Business-Critical Applications" NetForecast Report 5091. [online], http://www.netforecast.com/wp-content/uploads/2012/06/NFR5091SLAsforBusiness-CriticalApplications.pdf
Trigg, J. and Doulis, J. (2008) "Troubleshooting: What Can Go Wrong and How to Fix It", *Practical Guide to Clinical Computing- Systems: Design, Operations, and Infrastructure*, chapter 7, pp 105-128, 2008, Elsevier Inc, London.
Vacca, J.R. and Rudolph, K. (2010) *System Forensics, Investigation and Response*, Chapter 1, pp. 2-16. Sudbury, Mass.: Jones & Bartlett Learning
Worstall, T. (2012) "RBS/NatWest Computer Failure: Fully Explained", [online], http://www.forbes.com/sites/timworstall/2012/06/25/rbsnatwest-computer-failure-fully-explained/

Citation information

M. Bihina Bella, J. H. P. Eloff, and M. S. Olivier. "A Near-Miss Management System to Facilitate Forensic Investigation of Software Failures". In: Proceedings of the 13th European Conference on Cyber Warfare and Security. Ed. by A. Liaropoulos and G. Tsihrintzis. Academic Conferences, 2014, pp. 233–241