I. Mugarza, et al., Int. J. of Safety and Security Eng., Vol. 8, No. 1 (2018) 121–131

ANALYSIS OF EXISTING DYNAMIC SOFTWARE UPDATING TECHNIQUES FOR SAFE AND SECURE INDUSTRIAL CONTROL SYSTEMS

IMANOL MUGARZA[1], JORGE PARRA[1] & EDUARDO JACOB[2]
[1] IK4-Ikerlan Technology Research Centre, Dependable Embedded Systems Area.
[2] Faculty of Engineering, University of the Basque Country UPV/EHU.

ABSTRACT

Higher interconnectivity among devices, machines, the cloud and humans is envisioned in the actual trend of automation, also known as Industrial Internet of Things (IIoT). These industrial control systems, which may require high availability and/or safety related capabilities, are no longer isolated from the corporate environment or Internet. Software updates will be needed during the product life cycle, due to the long service life, the increasing number of security related vulnerabilities discovered on these industrial control systems and the high interconnectivity desired in IIoT. These updates aim at fixing all these security weaknesses, bugs and vulnerabilities that could appear, while the required safety integrity levels are ensured. Security-related concerns have just been addressed by the safety engineering community, because of the increasing number of cyber-attacks against safety-critical systems, such as Stuxnet. Moreover, system shut-downs caused by software updates could not be plausible when high availability is required. Typically, in order to perform the software update, the whole industrial process or the production is halted, so that the software upgrade is safely applied. However, this scenario might not be applied in critical infrastructures, such as nuclear or hydro-electrical power plants, where these production and service interruptions are not acceptable from the business and service point of view. This article presents an analysis of existing dynamic software updating techniques, which may be applied for safe and secure industrial control systems. These techniques aim at updating the running code, without the need of a halt and restart, increasing the availability of the industrial system.

Keywords: dynamic software updates, patches, safety, security, critical infrastructures

1 INTRODUCTION

In the actual trend of automation, also called Industrial Internet of Things (IIoT), high interconnectivity is requested. Sensors, actuators, processing units and people are connected to each other with the aim of conceiving a smart industrial system. Safety functions are carried out by some of the devices, so accidents and incidents which could impact on health are prevented. However, due to the high inter-connectivity among these industrial control systems, security concerns gain importance, especially for safety-critical systems, since these systems are no longer isolated from the open communications channels.

The first significant cyber-attack compromising safety and security, which targeted industrial control systems, was the Stuxnet computer virus. It was identified in 2010 and according to Ralph Langen it is possible to assume that the Natanz Uranium Enrichment plant in Iran was the only goal [1]. This malicious program targeted general-purpose and high-end Programmable Logic Controllers (PLC). Because of the increasing number of cyber-attacks against safety-critical systems, the safety engineering community has started to address those cyber-security threats, which can alter the proper functioning of safety-related systems [2, 3]. In this paper, the problem statement is first described and an analysis of existing dynamic software updating (DSU) techniques applicable to industrial systems then provided. A case study about nuclear reactor safety systems is finally given, where the need and use of DSU techniques is claimed.

© 2018 WIT Press, www.witpress.com
ISSN: 2041-9031 (paper format), ISSN: 2041-904X (online), http://www.witpress.com/journals
DOI: 10.2495/SAFE-V8-N1-121-131

2 PROBLEM STATEMENT

New security flaws are discovered every day. As reported by Kaspersky lab, the number of vulnerabilities in industrial control systems keeps growing. In 2015, 189 vulnerabilities were published, where 42% of them had medium severity and 49% were critical [4]. Likewise, the number of reported cyber-security incidents by the Industrial Control Systems Cyber Emergency Response Team (ICS-CERT) has grown since the inauguration of the team in 2009. Figure 1 depicts the total number of registered incidents per year, while the amount of reported incidents on some of the critical infrastructure sectors are also illustrated.

In order to keep the security level through the product service life, the software within the system needs to be updated periodically, as the same time as security flaws are discovered. However, a system shutdown due to a software update could not be plausible when high-availability is demanded. Examples of such scenario are a complex and demanding production line, a hydro-electrical power plant or a nuclear reactor control and protection system. In these cases, halting the control systems because of a software update may lead to compromise the safety properties of the process. This approach may not be practical from the business perspective. Moreover, if the shutdown process is not immediate, the system remains vulnerable until a safe power-off state is reached. This is the case when the safety-related system needs to actively maintain essential safety services. Cases of such scenarios are aircraft and nuclear reactor safety-critical systems. These services shall also be held up in case of a cyber-attack, since absence of process control may lead to hazardous events and accidents.

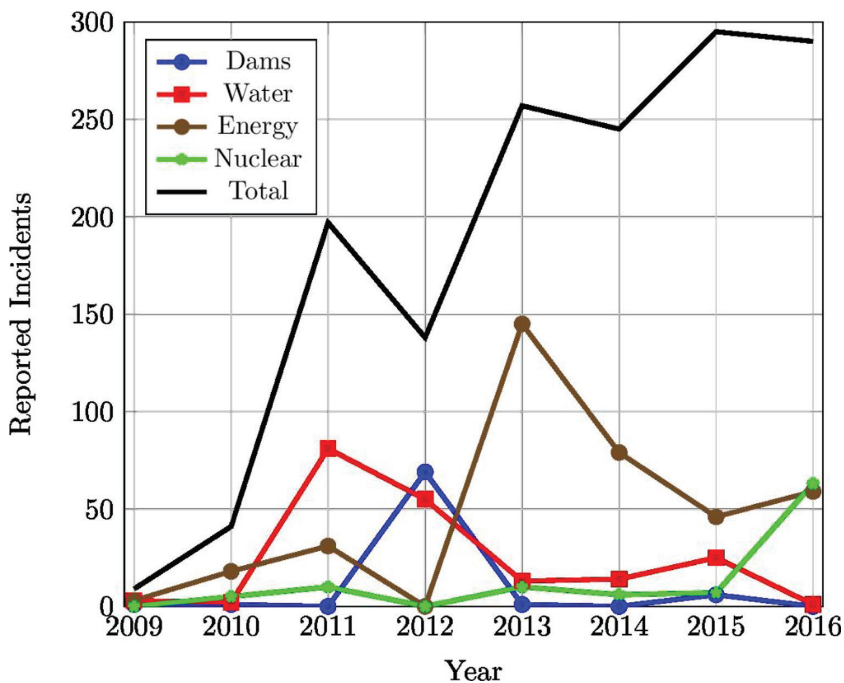

Figure 1: Reported cyber-security incidents in critical infrastructures by ICS-CERT.

3 DYNAMIC SOFTWARE UPDATING

Dynamic Software Updating (DSU) consists on updating a computer program while it is being executed without the need of a halt and restart. Redundant hardware is often used as an alternative to this approach to modify a running system on the fly. For this purpose, a secondary machine is employed. When a system update is desired, the new version of the code is loaded to the secondary system and the necessary state or information passed from the primary. After that, a role change between these machines is performed, where the primary machine is turned into the secondary, and the secondary into the primary one [5].

The execution of a computer program can be considered as a tuple (P, δ), where P is the program code and δ is the current program state. The current program state δ can include the state maintained by the operating system for the program P (such as file descriptors or open network connections), the heap, stack frames and program counters. In contrast, the program code P includes a set instructions executable by the system. The dynamic software updating mechanism transforms the actual running program (P, δ) to a new version (P', δ'). For this, the program code is first updated and the actual program state δ then transformed to $\delta^{'}$, so it is coherent with the new program code P' [5, 6].

A dynamic software update process consists of three aspects, which are: **code transformation**, **state transformation** and the **update point**. The **code transformation** refers to the process of updating the executable code, while the **state transformation** stands for the procedure of transforming the actual state of the program, so it understandable by the new program. Finally**, update point** refers to the execution instance where the software update occurs. The term update time is also employed, which denotes the time instance when the update takes place. Both of them refer to the same execution/time occurrence. Figure 2 shows the process time-line of a dynamic software updating process:

In case that the application is multi-threaded, the process becomes more challenging. Firstly, the executable code and the program state from each thread needs to be transformed. Secondly, it is required that all threads to reach an update point and wait to the others. Nevertheless, the possibility of unbounded delays of those threads when reaching an update point could lead to a service interruption from the application. Furthermore, a deadlock may occur, when a given thread is waiting for access given resource before reaching an update point, while another one has already reached it without releasing the resource. The application would then be halted [6, 7].

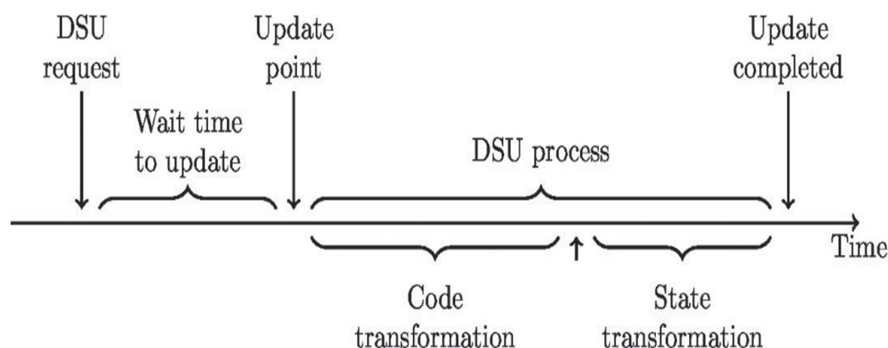

Figure 2: Dynamic Software Updating process time-line.

4 ANALYSIS OF EXISTING SYSTEMS

Over the years, many DSU systems have been proposed. These systems target many kind of applications, from real-time control purposes to servers and databases [8, 9]. Moreover, formal approaches have also been proposed, where DSU support at the programming language level is investigated. An underlying theory for programming languages which offer DSU features named Proteus is presented by *Stoyle* [10]. This theory is a calculus program and it was applied to the design and implementation of updatable C-programs. The most important challenges reside on how to address the unsafe features of C programming language, and how to design an efficient DSU feature.

Some of the actual programming languages already provide DSU support. Four programming languages were analysed by *de Pina* [11], which are: Common LISP, Smalltalk, Erlang and UpgradeJ. Even that these programming languages provide high-level DSU features; the developer is required to write the target applications on those languages. Consequently, the usage of these approaches is restricted for those applications developed from scratch in these languages. Incompatibility issues while integrating already existing libraries with the target application written in such programming language may also arise.

Another approach was followed by *P. Hosek* and *C. Cadar* [12, 13]. In this case, taking advantage of virtualization technologies, the new program version is executed in parallel with the old one, where their executions are synchronized. The aim of this technique is to maintain the stability of the old version while new features and bug fixes from the new version are also offered. A prototype called MX was implemented, where several applications where executed on multi-version mode. The system is composed upon three main components. The first one is the Static Executable Analyser (SEA), which performs a static analysis on both version binaries. The second one is the Multi-eXecution Monitor (MXM), where both versions are executed concurrently. Finally, the Runtime Execution Manipulator (REM) selects between the available behaviours and resynchronizes both versions in case of a divergence.

Commonly, the DSU systems provide a dynamic patch building system that generates a dynamic patch. It has to be noted that, as stated by *Hicks* [5], the concept of a dynamic patch differs from the regular static patch (*diff* and *patch* commands on UNIX), where it is described as the differences between sources of two program versions. A dynamic patch can consist on new updated executable code, new global variables, type/state transformers and additional meta-data. This information is then transferred to the DSU runtime system, where after receiving a DSU request, the DSU process is carried out [14]. These systems most often provide a set of tools, such as custom-compilers, source-to-source patch generators or source analysers in order to create dynamic patches. CIL [15] and LLVM [16] compilation, source analysis and transformation tools are commonly utilized [11].

In order to generate a dynamic patch, several tasks need to be accomplished by the dynamic patch building system [5, 11]. These activities are often monitored and/or aided by the developer. These tasks are:

1. Identification and syntactical comparison of the new program version against the old one(s).
2. Adaptation of the program (source and/or object) so it can be updatable.
3. Construct the type/state transformer functions (whenever possible).
4. DSU meta-data generation, where the information about current program version is defined.

A review of techniques, evaluation metrics and a survey of existing DSU systems have already been provided [8, 9]. These systems are categorized according to the used or characterized **code transformation**, **state transformation** and **update point** techniques and attributes. The DSU mechanisms are also assessed and discussed against the defined evaluation metrics. In this section, DSU systems which could be feasible for safe and secure industrial control applications are analysed. They have been divided into three categories, as similarly done by *de Pina* [11] and *Seifzadeh et al.* [8], which are:

- **Compiled application**-oriented DSU systems which target applications compiled to an executable binary. These objects are then executed natively.
- **Kernel**-oriented DSU systems which target the core of an operating system.
- **Real time**-oriented DSU systems, which have specifically been created for real-time, embedded or industrial control systems.

In the analysis carried out about DSU techniques by *de Pina* [11], compiled application and kernel groups were merged into a single one, since the core of an operating system can be considered a special purpose compiled application. Usually, the C programming language is used for this purpose. This programming language is usually also considered on compiled application-oriented DSU systems. The following Table 1 presents the analysed DSU systems. Note that the DSU systems are not ordered according to the date within each category. A similar arrangement used by *de Pina* [11] and *Seifzadeh et al.* [8] has been employed.

Table 1: Analysed DSU systems.

Name	Target	Date
DLpop	Compiled application	2001
Dynsec		2005
POLUS		2013
DynSec		2007
UpStare		2009
Ginseng		2008
Ekiden		2011
Kitsune		2012
Lucos	Kernel	2006
DynAMOS		2007
KSplice		2009
K42		2006
PROTEOS		2013
DURTS	Real-time	2004
EmbedDSU		2011
Gracioli		2014
EcoDSU		2008
Seif-Real		2009
Wahler		2009
FASA		2014

At the time of writing, and to the best of our knowledge, KSplice and its successors KPatch and KGraft mechanisms are only the DSU systems, which are applied and offered commercially for the Linux kernel. Live patching services are provided through these techniques, respectively, by Oracle Linux Premier Support, Red Hat and SUSE. The *livepatch* system was created in 2014 by Red Hat, unifying the KGraft and KPatch approaches, which is actually available at the mainstream kernel. In these DSU systems, only code transformations at function level are performed [17].

All the DSU systems targeting compiled applications perform dynamic updates in top of an UNIX-like operating system, usually GNU/Linux running on an x86 computer. Besides, UpStare has been tested on Linux 2.4–2.6 running on an i386 architecture computer, Solaris 5.10 running on a SPARC computer and MAC OSX running on a PowerPC computer [6, 18, 19]. However, none of these DSU systems provide real time features, which may be needed for an industrial control application. Except from DynSec, all the DSU systems targeting compiled applications present a dynamic patches building toolchain. In case of DSU systems designed for operating system kernels, LUCOS [20], DynAMOS [21] and KSplice [17, 22] (and its successors KGraph and KPatch) target the Linux kernel while K42 is the operating system kernel itself, designed to provide customizability, scalability and maintainability [23–26]. PROTEOS [27, 28] is designed for the MINIX 3 operating system, which is compatible with the POSIX interface [29]. In this DSU system, where process level upgrades are performed, automatic state transfers are provided. PROTEOS is executed on x86 computers and according to the author, the proposed technique can be easily applied to other operating systems, for example: L4, Integrity or QNX.

DURTS [30] is the only real-time DSU targeting a UNIX-like operating system, specifically, the RT-Mach operating system. Other specific operating systems where used on EmbedDSU [31] and Gracioli [32]. While EcoDSU [33] works purely on bare metal, without the support of an operating system, Wahler [34] and FASA (Future Automation System Architecture) [35–38] are operating system agnostics. This means that any operating system can be used. Nevertheless, they shall be POSIX-compatible.

None of the analysed DSU system provides a safety compliant solution. As stated by *Wahler et al.* [39], the biggest challenge for updating the software of real-time systems is that they often need to be certified according to standards such as the IEC 61508 [40]. If the software of such system is updated, the system as a whole must be re-certified before it can be used in the field. Most of the DSU systems perform software upgrades on top of an UNIX-like operating systems, which are not advised for safety-related systems, even that initiatives to evaluate and assess the Linux kernel against safety standards exist [41, 42].

Wahler, FASA and PROTEOS systems are the closest proposed solutions towards a safety-compliant DSU technique, since as stated by the authors, the described techniques are compatible with any POSIX-compliant operating system, where a safety certified one could be chosen. In case of PROTEOS, the core of the operating system needs to be modified. Consequently, after the changes, it shall be re-certified. Nevertheless, Wahler and FASA take advantage of the operating system utilities, so the DSU is transparent to the underlying operating system. The application, libraries and the dynamic updating mechanism running on top of the operating system shall be then certified.

One of the requirements for the safety assessment is the schedulability analysis, where it is ensured that tasks are executed and completed according to their timing requirements. This property was examined from the DSU perspective in DURTS [30] and Seif-Real [43]. Just the main idea was presented in Seif-Real, without specifying any of the **code transformation**, **state transformation** and **update point** aspects.

5 CASE STUDY

In 2010, the Stuxnet virus demonstrated that nuclear facilities are also exposed to cyber-security threats [44]. As shown in Fig. 1, a notorious increment of cyber-security incidents in nuclear sector (from 7 to 63) was registered last year by the Industrial Control Systems Cyber Emergency Response Team (ICS-CERT). Thus, due to the increasing number of vulnerabilities on industrial control systems and long operational life of these installations software updates are necessary, so security weaknesses, bugs and vulnerabilities are resolved. In this case study, nuclear reactor safety systems are evaluated as far as security software updates are concerned. We claim that due to the complexity and criticality of nuclear processes, dynamic software updates are necessary.

5.1 Nuclear reactor safety systems

When an anomalous behavior or unsafe conditions of the reactor is detected, the Reactor Protection System (RPS) shall effectuate an emergency shutdown. The nuclear reaction chain is then broken through the insertion of control rods. In presence of failure of the RPS, a secondary shutdown mechanism, such as boron injection, is used. At this point, even the nuclear reactor is powered off, decay heat is still produced as an effect of radiation [45]. At the time instance that the reactor is shutdown, the decay heat is approximately the 6.5% of the previous reactor power, assuming that a steady and long power history was accomplished. The following Fig. 3 shows the decay heat decrease after one year of reactor operation from the time instance in which it was taken down [46].

Although the nuclear reactor is shutdown, because of the decay heat, essential safety functions have to be carried out to cool down the reactor. If the decay heat is not removed, unsafe conditions of the reactor can be reached, which could lead to a nuclear disaster [44, 45]. For example, a partial meltdown of the unit 2 on the Three-Mile Island took place after reactor shutdown because equipment failure and operator fault. On the contrary, in Fukushima, even the reactors were turned off after the earthquake, the tsunami incapacitated all the emergency

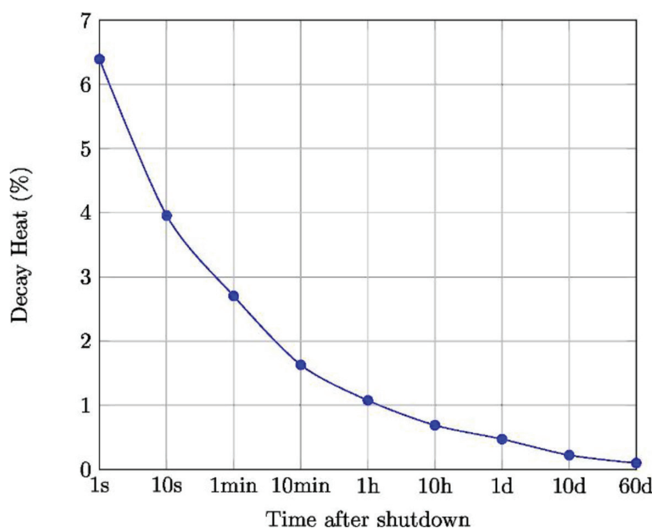

Figure 3: Nuclear reactor decay heat after 12 months of operation

power supply generators necessary for the reactor cooling systems. In consequence, Units 1, 2 and 3 were meltdown [47].

5.2 Security updates

If security weaknesses, bugs or vulnerabilities are discovered on the reactor safety systems, a decision of upgrading or not the affected equipment has to be taken. On the one hand, if the protection system is decided to be updated, the whole nuclear process shall be first halted, so the software upgrade is safely applied. This might not be conceivable from the business and service point of view. In contrast, the safety integrity level would be compromised if the update is applied while operation of the nuclear reactor, because the power-off of the safety system is required. On the other hand, if the decision of not updating the system is taken, the system would remain attackable. As the system is no longer secure, it is not safe either. Thus, the only approach which does not compromise safety is to halt the whole process, which it may not be acceptable from the service and business perspective. Note that, nuclear reactors are usually run without interruption for one or two years once the fuel is allocated.

The scenario get worse in case a cyber-attack is detected while the reactor safety systems continue being vulnerable. If the cyber-attack compromises the reactor protection systems, a nuclear accident might be unavoidable. On the assumption that the attack is identified in time, or provoked anomalies detected, the nuclear reactor would be powered-off by the reactor protection systems. In order to prevent any kind of software infection or other similar consequences, these systems could also be halted afterwards. However, even the nuclear reactor is shutdown, other safety systems shall maintain essential services, such as cooling of the reactor or the backup power supply. These safety systems are then totally at the mercy of the cyber-attack, since an appropriate operation of these systems is necessary in order to have under control the reactor. In absence of control, a nuclear disaster could be imminent. The attackers could also take down on purpose these safety-critical systems. Only those systems enabled with dynamic updating features could be upgraded in situ, so consequences of the cyber-attack could be mitigated. These techniques give the possibility of solving these cyber-security issues without the interruption of the service and safety properties.

6 CONCLUSIONS

Due to the increasing number of vulnerabilities in industrial control systems, software updates are needed. However, a system shutdown due to a software update is not acceptable for those applications where zero downtime is required, for example aircraft, hydro-electrical or nuclear safety-critical systems. In this article, an analysis of existing dynamic software updating techniques for safe and secure industrial control systems is given. These mechanisms aim at upgrading the actual running program while it is executing without of need of halt and restart.

As claimed by *Wahler et al.* [39], software within safety-critical systems has to usually be certified according to safety standards. A high safety confidence level is also required for the software development and supporting tools. To the best of our knowledge, no applicable safety-compliant DSU mechanism exists, but Wahler, FASA and PROTEOS systems are the closest proposed solutions towards a safety-compliant one. Nevertheless, further work is necessary, especially to ensure and validate the integrity and correctness of those software updates. Consequently, at the time of writing, the only approach to update safety-critical systems on-the-fly is redundancy.

REFERENCES

[1] Langner, R., Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security Privacy,* **9**(3), pp. 49–51, 2011.
https://doi.org/10.1109/msp.2011.67

[2] Paul, S. & Rioux, L., Over 20 years of research in cybersecurity and safety engineering: a short bibliography. *Safety and Security Engineering VI.*
https://doi.org/10.2495/safe150291

[3] Paul, S., On the meaning of security for safety (S4S). *WIT Transactions on The Built Environment,* **151**, pp. 379–389, 2015.

[4] K. S. Intelligence, "Industrial cybersecurity threat landscape," ed, 2016.

[5] Hicks, M., Moore, J.T. & Nettles, S., Dynamic software updating. *ACM*, **36**(5), pp. 13–23, 2001.

[6] Makris, K., *Whole-program dynamic software updating*. Arizona State University, Tempe, AZ, 2009.

[7] Hayden, C.M., *Clear, correct, and efficient dynamic software updates*, 2012.

[8] Seifzadeh, H., Abolhassani, H. & Moshkenani, M.S., A survey of dynamic software updating. *Journal of Software: Evolution and Process,* **25**(5), pp. 535–568, 2013.
https://doi.org/10.1002/smr.1556

[9] Miedes, E. & Munoz-Escoi, F.D., *Dynamic software update,* Technical Report ITI-SIDI-2012/0042012.

[10] Stoyle, G., A theory of dynamic software updates, 2007.

[11] de Pina, L.G.G., *Practical dynamic software updating*. INSTITUTO SUPERIOR TECNICO, Lisbon, Portugal, 2016.

[12] Cadar, C. & Hosek, P., Multi-version software updates. In *Proceedings of the 4th International Workshop on Hot Topics in Software Upgrades*, pp. 36–40, 2012.

[13] Hosek, P. & Cadar, C., Safe software updates via multi-version execution. *Proceedings of the 2013 International Conference on Software Engineering*, pp. 612–621, 2013.

[14] Neamtiu, I., Hicks, M., Stoyle, G. & Oriol, M., Practical dynamic software updating for C. *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, pp. 72–83, 2006.

[15] Necula, G.C., McPeak, S., Rahul, S.P. & Weimer, W., CIL: Intermediate language and tools for analysis and transformation of C programs. *International Conference on Compiler Construction*, pp. 213–228, 2002.

[16] Lattner, C. & Adve, V., LLVM: A compilation framework for lifelong program analysis & transformation. *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, p. 75, 2004.

[17] Binnie, C., Zero downtime Linux. In *Practical Linux topics*: Springer, pp. 33–39, 2016.

[18] Makris, K. & Bazzi, R.A., Immediate multi-threaded dynamic software updates using stack reconstruction. *USENIX Annual Technical Conference*, San Diego, CA, 2009.

[19] Makris, K., Upstare manual," ed, 2012.

[20] Chen, H., Chen, R., Zhang, F., Zang, B. & Yew, P.-C., Live updating operating systems using virtualization. *Proceedings of the 2nd international conference on Virtual execution environments*, Ottawa, Ontario, pp. 35–44, 2006.

[21] Makris, K. & Ryu, K.D., Dynamic and adaptive updates of non-quiescent subsystems in commodity operating system kernels. *ACM SIGOPS Operating Systems Review,* **41**(3), pp. 327–340, 2007.
https://doi.org/10.1145/1272998.1273031

[22] Arnold, J. & Kaashoek, M.F., Ksplice: Automatic rebootless kernel updates. *Proceedings of the 4th ACM European conference on Computer systems*, Nuremberg, Germany, pp. 187–198, 2009.

[23] Krieger, O., Mergen, M., Waterland, A., Uhlig, V., Auslander, M., Rosenburg, B., Wisniewski, R.W., Xenidis, J., Da Silva, D., Ostrowski, M., Appavoo, J. & Butrico, M., K42: Building a complete operating system. *ACM SIGOPS Operating Systems Review,* **40**(4), pp. 133–145, 2006.
https://doi.org/10.1145/1218063.1217949

[24] Baumann, A., Appavoo, J., Da Silva, D., Krieger, O. & Wisniewski, R.W., Improving operating system availability with dynamic update. *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure*, Boston, MA, pp. 21–27, 2004.

[25] Baumann, A., Dynamic update for operating systems. *Doctor of Philosophy, School of Computer Science and Engineering, The University of New South Wales,* vol. 112, 2007.

[26] Baumann, A., *et al.*, Providing Dynamic Update in an Operating System. *USENIX Annual Technical Conference, General Track*, pp. 279–291, 2005.

[27] Giuffrida, C., Kuijsten, A. & Tanenbaum, A.S., Safe and automatic live update for operating systems. *ACM SIGARCH Computer Architecture News*, **41**, pp. 279–292, 2013.
https://doi.org/10.1145/2499368.2451147

[28] Giuffrida, C. and others, *Safe and Automatic Live Update*. VU University Amsterdam, 2014.

[29] Tanenbaum, A.S. & Woodhull, A.S., *Operating Systems Design and Implementation*, 3rd edn., Prentice-Hall, Inc.: Upper Saddle River, NJ, 2005.

[30] Montgomery, J., A model for updating real-time applications. *Real-Time Systems,* **27**(2), pp. 169–189, 2004.
https://doi.org/10.1023/b:time.0000027932.11280.3c

[31] Noubissi, A.C., Iguchi-Cartigny, J. & Lanet, J-L., Hot updates for Java based smart cards. *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference*, pp. 168–173, 2011.

[32] Gracioli, G. & Fröhlich, A.A., An operating system infrastructure for remote code update in deeply embedded systems. *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades*, New York, NY, 2008, p. 3.

[33] Kang, S., Chun, I. & Kim, W., Dynamic software updating for cyber-physical systems. *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, pp. 1–3, 2014.

[34] Wahler, M., Richter, S. & Oriol, M., Dynamic software updates for real-time systems. *Proceedings of the 2nd International Workshop on Hot Topics in Software Upgrades*, Orlando, FL, p. 2, 2009.

[35] M. Oriol, Wahler, M., Steiger, R., Stoeter, S., Vardar, E., Koziolek, H. & Kumar, A., FASA: a scalable software framework for distributed control systems. *Proceedings of the 3rd international ACM SIGSOFT symposium on Architecting Critical Systems*, pp. 51–60, 2012.

[36] Wahler, M., Oriol, M. & Monot, A., Real-time multi-core components for cyber-physical systems. *2015 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*, pp. 37–42, 2015.

[37] Monot, A., Oriol, M., Schneider, C. & Wahler, M., Modern software architecture for embedded real-time devices: high value, little overhead. *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 201–210, 2016.

[38] Wahler, M., Gamer, T., Kumar, A. & Oriol, M., FASA: A software architecture and runtime framework for flexible distributed automation systems. *Journal of Systems Architecture,* **61**(2), pp. 82–111, 2015.
https://doi.org/10.1016/j.sysarc.2015.01.002

[39] Wahler, M., Richter, S., Kumar, S. & Oriol, M., Non-disruptive large-scale component updates for real-time controllers. *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference*, pp. 174–178, 2011.

[40] I. E. Commission and others, "Functional safety of electrical/electronic/programmable electronic safety related systems," *IEC 61508,* 2000.

[41] Pierce, R.H., *Preliminary assessment of Linux for safety related systems*. HSE Books, 2002.

[42] Mc Guire, N., Linux for safety critical systems in IEC 61508 Context. *Proceedings of the Ninth Real-Time Linux Workshop in Linz*, 2007.

[43] Seifzadeh, H., Kazem, A.A.P., Kargahi, M. & Movaghar, A., A method for dynamic software updating in real-time systems. *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, pp. 34–38, 2009.

[44] Kesler, B., The vulnerability of nuclear facilities to cyber attack. *Strategic Insights,* **10**(1), pp. 15–25, 2011.

[45] Thomson, J., *High Integrity Systems and Safety Management in Hazardous Industries*. Butterworth-Heinemann, 2015.

[46] Garland, W.J., Decay heat estimates for MNR. *McMaster Nuclear Reactor, McMaster University, Ontario,* 1999.

[47] Holt, M., Campbell, R.J. & Nikitin, M.B., *Fukushima nuclear disaster*. Congressional Research Service, 2012.