BULLETIN of Cherkasy State Technological University

Journal homepage: https://bulletin-chstu.com.ua/en Vol. 29 No. 1. 2024

UDC 004.49
DOI: 10.62660/bcstu/1.2024.36

On specifics of adaptive logging method implementation

Illia Suprunenko*
Postgraduate Student
Cherkasy State Technological University
18006, 460 Shevchenko Blvd., Cherkasy, Ukraine
https://orcid.org/0000-0002-1188-4804

Volodymyr Rudnytskyi
Doctor of Technical Sciences, Professor, Chief Researcher
Cherkasy State Technological University
18006, 460 Shevchenko Blvd., Cherkasy, Ukraine
State Scientific Research Institute of Armament and Military Equipment Testing and Certification
18000, 164 Vyacheslav Chornovil Str., Cherkasy, Ukraine
https://orcid.org/0000-0003-3473-7433

Abstract. Relevancy of this work is based on the fact that having an understanding of why given code behaves the way it does, both during normal execution and when encountering erroneous states, is an invaluable part of a good software design. As software systems become more complex, the demand for solutions, that can give deeper insight into code execution, remains high. The goal of this work is to formalize a software tool able to provide better observability of a program. Main methods used are: analysis of common approaches such as monitoring and logging, formalization of main components and modeling of an example implementation based on the Singleton software pattern. As a result, "severity only" based logging was analysed and core parts of "adaptive logging method" were described in a similar manner. There are two distinct features of this method: log tagging and subsequent introduction of a configuration schema that is capable of adapting to changing requirements during software program execution. Systems utilizing such approach gain the ability to extract more precise information about execution flow and also can focus on particular components that might behave incorrectly. As this switch is designed to happen without restarting the observed program, it should be possible to debug and investigate some issues without the need to try and reproduce from scratch the state of an environment where those have occurred. An example of formal description based on the Singleton software pattern is also presented, describing necessary methods and their signatures required to set up a basic variant of an adaptive logging method. This approach could be utilized by a variety of different applications and programming languages as it is developed in general terms and all required abstractions should be present in multiple environments

Keywords: cyber attacks; information security; observability; debugging; adaptive approach; application model

Article's History: Received: 06.09.2023; Revised: 11.11.2023; Accepted: 18.03.2024.

INTRODUCTION

Software systems are deeply connected with various parts of human lives and every year more and more areas such as education, medicine, commerce, science and others become increasingly computerized. It allows being more productive and makes everyday life easier. In order to satisfy high demands of such systems, software programs grow both in scale and complexity. This also results in growing number of threats, endangering different aspects of human lives. The relevance of this research is based on the fact that it is equally

Suggested Citation:
Suprunenko, I., & Rudnytskyi, V. (2024). On specifics of adaptive logging method implementation. *Bulletin of Cherkasy State Technological University*, 29(1), 36-42. doi: 10.62660/bcstu/1.2024.36.

*Corresponding author

important to create efficient, correct, scalable and performant programs and to protect main aspects of information security – integrity, availability, confidentiality – while also providing sufficient level of observability in a software system.

Studies have shown that information technologies, and the Internet in particular, considerably affect the life of an individual. Total number of Internet users in 2023 was a little bit over 5 billion people and that number grew by 98 million new users compared to previous year, with more than 92% of users accessing it using hand held devices like smartphones (Digital…, 2023). It is evident that information security and its importance are widely recognized: spendings on cybersecurity solutions by various companies increased from 34 billion dollars in 2017 to a projected 42 billion dollars in 2020 (Alenezi *et al.*, 2020). Still security issues remain quite challenging, as software threats come in different shapes and forms. Social engineering, for example, is a very powerful way of gaining information about employees and exploiting their weaknesses. Even though these attack vectors were known for years, it is still quite challenging to handle them effectively (Corradini, 2020). Attackers can even target such sensitive parts of life as health care, causing huge monetary losses and endangering individuals (Joyce *et al.*, 2021). Worldwide crises, such as Covid-19 pandemic, are also actively exploited by attackers, who create malicious software, send ransomware, etc, using vulnerable state people found themselves in to their advantage (Yadav, 2021). From February to March 2020 the number of recorded spam emails was 220 times higher than previously. Some of the most common cybersecurity threats were: Distributed Denial of Service (DDOS) attacks, malicious websites and domains, ransomware, mobile threats and spam emails (Khan *et al.*, 2020).

Solutions to some of the problems are widely known, and a common one to avoid compromising privacy of data on the Internet is to use virtual private network (VPN) technology, which allows for information exchange to be protected using assymetric cryptographic encryption algorithm (Zhylin *et al.*, 2020). According to data recorded during initial lockdowns in Poland, the usage of Open VPN compared to pre-lockdown periods almost doubled and that was the first and straightforward consequence of the measure's governments took to prevent spread of the virus (Karpowicz, 2021). Such services aided a lot in mitigating issues that appeared as a lot of employees were forced to work from home (Bispham *et al.*, 2021). They can also help in common scenarios such as accessing sensitive data from public Wi-Fi networks, but even high level of awareness can sometimes be not enough (Sombatruang *et al.*, 2020). Control is another important aspect in programming. This research is focused on one part of control in software systems – observability, which can be achieved using technique called "logging". Application logging can be described as a process of recording events that happen during application's lifecycle (Application logging…, n.d.). In order to fully benefit from any implementation, it is necessary to remember about the costs of logging and avoid exposing unnecessary information (Li *et al.*, 2021).

The main goal of this work was to present a method that should help developers gain a deeper level of observability in their systems, while allowing to be accurate and comprehensive in finding software problems. The main problem of this research is related to the fact that standard "severity only" based approach to logging can sometimes be really hard to leverage in order to fix bugs in a timely manner. The practical novelty of the results is presented as a formal description of a software implementation for the proposed solution.

MATERIALS AND METHODS

There are three main methods used in this research: analysis, formalization and implementation. First of all, analysis of a general "severity only" approach to logging is conducted. Its formal signature is presented and some drawbacks are emphasized. Based on those drawbacks, requirements and signatures for adaptive logging method are created, describing main components of common practical implementations. Usage of the Singleton software pattern is rationalized and example implementation is constructed in general terms using Unified Modeling Language (UML) diagram. Different parts of practical implementation are purposefully designed in a decoupled way, without any bindings to a specific programming language or execution environment, in order to make adaptive logging method as flexible and reusable as possible. Resulting description can be used by developers as a starting point in implementing this method in specific programs and software solutions.

One of the most widely used methods in practice of tracking application state through logging can be described as a "severity only" one. Severity in this case basically means the level of importance of a log message. The RFC 5424 – Syslog Protocol (RFC – Request for Comments) states that severity levels must be in the range of 0 to 7 inclusive (lower values correspond to more dangerous situations) (Gerhards, 2009). Based on those practical implementations, such as Winston logger library (Winston, 2024), define their general "log" method (1) as such that can accept two parameters: severity level literal and log message contents (most commonly a string value).

$$f_{log} = f(Sev, M), \tag{1}$$

where Sev – a severity of current log invocation; M – a message.

Then these reporting messages might end up in standard output stream, on local or even on remote file systems, aggregated in huge log files that require careful processing because of their size. However, there are some solutions, like AMiner – open-source tool

that enables fast log parsing, analysis and alerting (Landauer *et al.*, 2023) or even some techniques that can greatly aid in filtering and processing. For example, H.M. Marin-Castro & E. Tello-Leal (2021) showed that trace-clustering and trace/event level filtering were the most widely used preprocessing techniques as those are easy to implement and can deal with noise and incompleteness in an adequate manner. Sometimes it is much more desirable to filter out unneeded data during generation step rather than during log examination phase. This is also important from performance perspective as naturally being a side effect to the system logging introduces some overhead that can limit effectiveness and impact users of a software system.

In order to allow for generation phase filtering, logging mechanism has to allow access to three main components: ability to mark particular log invocations with descriptive identifiers ("tags"), set up proper reinitialization mechanism that would allow to change configuration of tags and provide a robust and exhaustive configuration method that is used together with reinitialization function when composing next state for the logging software solution.

RESULTS AND DISCUSSION

The starting point of the formalization procedure is to describe adaptive log method (2) signature. Its parameters list starts similarly to the one from "severity based" log method (1), but has an additional parameter.

$$f_{log\ adp} = f(Sev, M, T_{incl}), \quad (2)$$

where Sev – a severity of current log invocation; M – a message; T_{incl} – a set of tags that describe current invocation.

As a result, it is possible to uniquely identify and group messages, while being more declarative compared to some method of project wide convention of prefixing or marking log message strings with some agreed identifiers. By adding the third parameter to log method signature, more consistent behavior can be enforced and this improvement can be taken a step further, making it truly adaptive. Tagging allows to group and identify particular log invocations, adding the ability to run some computations and decision making based on tags. The runtime environment needs access to some form of configuration mechanism in order to specify which invocations to execute and which to skip. This method should combine severity-based mechanism, but also add something that would allow targeting specific log tags (3). Signature for this config initialization method is:

$$f_{init} = f(Sev, C), \quad (3)$$

where Sev – the lowest severity level runtime should report; C – special configuration object that maps tags to their corresponding action, include or exclude.

The contents of the C configuration object (4) should be constructed in such a way that would: allow to have the ability to explicitly exclude specific log tags, which would result in runtime skipping those from output; add means to explicitly include certain tags; provide some way to combine these two in a relation similar to logical "AND", which would mean that only those logging statements that satisfy both operands of this operator, should be included in final output; for complex and multipurpose setups (when it is important to gain insight into different parts of running software during the same execution cycle) there should also be some way to combine tags, joined by "AND", into structure that would be similar to logical "OR" operation.

$$C = T_{11}^{mod}\ \&\&\ T_{12}^{mod}\ \&\&\ldots\|\|\ T_{21}^{mod}\ \&\&\ T_{22}^{mod}\ \&\&\ldots\|\|\ldots, \quad (4)$$

where T_{ij} – a particular tag; mod (modifier) – "include" or "exclude"; && – equivalent of logical "AND" operator; || – equivalent of logical "OR" operator.

By combining the ability to mark logger calls and configuring the desired set of tags to either include or exclude, this mechanism can greatly lower the verbosity of reporting while also reducing load and resource consumption of software. This is essentially what can be called "adaptive logging method" because of its ability to specify the intent more clearly and in a way "adapt" to changed requirements.

As application logging is generally a system wide concern, it's fairly reasonable to expect to be able to utilize corresponding software in different parts of code. It also needs the ability to apply reinitialization method (3) in a portable manner where it can be called once and be altered in all places where it's used. The Singleton programming pattern is one possible solution that is able to cover these requirements. With it, it is possible to have centralized piece of software that can be used in any part of a program and overriding logging configuration in accordance to changed requirements happens in a very natural form. It is worth mentioning that whole idea of adaptive logging serves as a wrapper for general logging mechanisms and as such this research won't focus on any logging related tasks (as modeling and developing those correctly would possibly require solving issues like working with different formats, utilizing different transports and protocols for log transmission, handling synchronous and asynchronous processing, etc., and is out of the scope of this discussion). Instead, it is expected for existing, well tested solutions (like Winston logger) to be leveraged, using a wrapper around those to add adaptive logging functionality.

The instance of a Singleton would first need to have logging library or framework enclosed in one of its member properties (it might even be completely private to enforce correct usage of dedicated methods). The interaction with this framework should happen through a public method which basically implements adaptive

log method signature (2). This is going to be the main reference point in different parts of the program: if it is desired to capture some details about execution process, it gets invoked with corresponding severity level and desired tags are passed to it for identification (with some text-based description). A method that allows overriding current configuration (called, for example, "init") and is basically an implementation of equation (3) should be another part of public Singleton interface.

To better visualize intended structure of an example implementation a common UML class notation is used, where class attribute corresponds to logging library member property and class operations are essentially two public methods required for proper functionality. This example does not limit class attribute by using particular type signatures as different logging frameworks might be presented having different public interfaces, but for public methods signatures are provided for both parameters and return types (note, that "void" in context of function return type just means that function is not expected to return anything after its execution).

It's also worth noting that described class instance is designed to be as general and programming language agnostic as possible and used type signatures should have similar concepts in multiple languages. As such, "List" basically describes some form of an array of items, "Tuple" represents a structure with exactly two elements and for "modifier" it is expected to have exactly one of two possible literal values. This structure presented in Figure 1 should present a good starting point for anyone trying to implement "adaptive logging method" in their own workflow. As mentioned before, this definition is expected to be fairly portable to many existing implementations and workflows based on its general shape and decoupled type signatures.

Adaptive logging Singleton
- logLibrary (e.g. Winston logger)
+ **log**(*severity*: string, *tags*: List< string >, *message*: string): void + **init**(*severity*: string, *config*: C): void, where + *C*: List< OrSegment >; + *OrSegment*: List< AndSegment >; + *AndSegment*: List< TagDef >; + *TagDef*: Tuple< string, modifier >; + *modifier*: "include" or "exclude";

Figure 1. Adaptive logging Singleton structure
Source: compiled by the authors

At this point all main results, such as formal definitions and example formal implementation using Singleton software pattern, are presented. It is worth discussing where these results fit compared to similar or related solutions. Observability based on logging is one of commonly used approaches, monitoring, which can be characterized as the task of assessing the health of a system using predefined metrics that are collected by runtime environment and can then be analysed, is the other one (Observability…, 2022). It deals mainly with predefined metrics, such as memory usage, central processing unit (CPU) utilization, throughput, number of input/output (IO) operations, etc. Monitoring is not limited by those, and, for instance, it is possible to use natural language processing for extracting key data from the software monitoring data in order to predict possible defects (Wang *et al.*, 2021). In combination with, for example, neural network technologies, it is possible to develop complex software monitoring and early warning systems for such sophisticated fields as Internet-of-Things (Ma *et al.*, 2022). In contrast, logging allows to report on multiple different attributes and is not really limited by some common metrics, which is why being able to adapt to changed requirements is a significant capability to have in a software system. As for natural language processing and neural networks usage, adaptive logging method in its current form aims to reduce need for those (as it should produce less unwanted noise and relies more on manual input), but as software systems grow, it should certainly be possible to employ those techniques improving initial ideas and, for example, proactively alter reporting precision using some form of neural network-based solution.

Testing is another common solution for control related issues in programming. It is defined as a "process of assessing the functionality of a software program" and is generally accomplished through writing special test suits and test cases that aim to provide a system, or a part of it, with some inputs and then validate obtained results, although testing might not be entirely sufficient method for preventing software failures (Chioteli *et al.*, 2022). Some terminology still might require refinement to better correspond to the real-life scenarios and requirements instead of using formal approaches that might not always work as expected (Trautsch *et al.*, 2020). It provides a basic development technique that allows the developer to ensure that the code works in a more reliable and controlled manner. In contrast to logging, and adaptive logging in particular, main difference would be that testing relies on predefined and prepared sets of inputs, while logging is more about runtime examination of program's behavior. However, those should not be treated as replacements of one another and instead the best possible results in terms of more reliable software can be achieved having both and improving those synchronously: issues detected using logging might become new test suits and bottlenecks discovered while running different types of testing can tell which parts of the system are error-prone and require more detailed reporting.

Adaptive logging approach is different from, for example, the technique presented in research conducted by H.M. Kabamba *et al.* (2023). While their focus was mainly to introduce a solution that can be applied transparently to a software system, with limited intervention from an application developer, for some

scenarios more suitable solution would be to empower software engineer to be as precise as they need to be and limit not the amount of work needed to setup appropriate reporting, but rather amount of effort needed to process results. On the other hand, approaches like log slicing, presented in study by J.H. Dawes *et al*. (2023), work in a similar way and even solve related problems, for example log identification. Although their way of determining the related context given a particular log invocation looks interesting, adaptive logging emphasizes that it's best to solve those aspects during log generation phase, rather than during processing phase afterwards, as this can help in both limiting the amount of information to process and also in limiting load that runtime machine needs to deal with.

J. Torres Martínez *et al*. (2019) noted that different cybersecurity issues such as intrusion detection, attack classification, etc. can be tackled using machine learning approaches. Possible application domains of such techniques are: filtering spam in email messaging, blogs in social networks, images and videos, intrusion detection in a form of Denial of Service (DOS) attacks, probe attacks, "Remote to Local" and "User to Root" attacks, with very impressive results, such as 97% accuracy of DOS attacks detection using techniques such as Decision Tree or Neural Networks, as was mentioned by K. Shaukat *et al*. (2020). In a way, the use of this technology can reduce the need for logging as an observability tool (as several problem domains can be covered without resorting to manual interaction, for example in DOS detection), and in case of adaptive logging this might significantly reduce the need for a human input, rendering adaptability benefits somewhat redundant. But despite such promising results, broad application of machine learning systems could be problematic based on issues related to expensive training and narrow application fields for particular models, according to F. Liang *et al*. (2019), and so, at least for the time those approaches evolve, it is still very important to have "hands on" control over software systems, which adaptive logging aims to provide.

Adaptive logging formalization involves defining a method with a signature comprising a severity parameter, a message, and tags for unique message identification and grouping. This method combines severity-based logging with tag targeting through a configuration mechanism to specify which log invocations to execute or skip, aiming to enhance logging clarity and adaptability. The approach employs the Singleton programming pattern for portable reinitialization and acts as a wrapper for existing logging mechanisms such as the Winston logger. A Singleton for adaptive logging includes a logging library property, accessible through a public method with an adaptive log signature, and allows configuration overriding through a method like "init". Adaptive logging provides a flexible alternative to traditional monitoring and testing methods, complementing monitoring's focus on predefined metrics and testing's reliance on predefined inputs, thereby improving software reliability when used in conjunction. It enables developers to efficiently manage reporting precision and remains valuable alongside machine learning solutions for addressing cybersecurity issues.

CONCLUSIONS

In this work a deeper look at control issues in a software system is presented (mainly in a form of an observability aspect of information security). The research is focused on an aspect of application logging, and a general "severity only" based approach is analysed. Its main drawback is concluded to be an insufficient degree of flexibility and as a result inability to adapt to changes in reporting requirements. The solution to this problem is presented in a form of "adaptive logging method", that can be provided with the same parameters (severity and a message), but also utilizes a mechanism of tagging, which then allows to set up a specific architecture that should be capable of solving adaptivity issue. All of the necessary parts are formalized and presented as a general function signature (which should allow for cross platform and cross language implementations) and an example implementation based on the Singleton software pattern is presented using a Unified Modeling Language class notation. The resulting solution allows both to group related log calls using tags and to apply filter-like capability at the log generation phase, which makes it more precise than common logging approach and also allows to tone down performance impact of a logging solution by targeting very specific parts of an application.

Among topics for further research, several present a particular interest, for example selecting an appropriate subscription-based change propagation mechanism in order to adapt to changed requirements in a timely manner; improving adaptability in terms of not only "when" to log something, but also "what" to log, adding some layer of control on top of a tag-based mechanism. Another topic that could be covered is exploring possible integration with modern application orchestration mechanisms, that could potentially allow to reroute loads from erroneous services and isolate those for further investigation, as well as utilize machine learning approaches to gather foundational materials that could help in establishing some basic logging scenarios, created using the information extracted from bug reports of the frameworks and libraries used in a project.

ACKNOWLEDGEMENTS

None.

CONFLICT OF INTEREST

None.

REFERENCES

[1] Alenezi, M.N., Alabdulrazzaq, H., Alshaher, A.A., & Alkharang, M.M. (2020). Evolution of malware threats and techniques: A review. *International Journal of Communication Networks and Information Security (IJCNIS)*, 12(3), 326-337. doi: 10.17762/ijcnis.v12i3.4723.

[2] Application logging: Definition, examples, and best practices. (n.d.). Retrieved from https://coralogix.com/guides/application-performance-monitoring/application-logging-best-practices/.

[3] Bispham, M., Creese, S., Dutton, W.H., Esteve-González, P., & Goldsmith, M. (2021). Cybersecurity in working from home: An exploratory study. In *TPRC49: The 49th research conference on communication, information and internet policy* (pp. 1-43). Oxford: University of Oxford. doi: 10.2139/ssrn.3897380.

[4] Chioteli, E., Ioannis, B., & Diomidis, S. (2022). Does unit-tested code crash? A case study of eclipse. In *PCI '21: Proceedings of the 25th pan-hellenic conference on informatics* (pp. 260-264). New York: Association for Computing Machinery. doi: 10.1145/3503823.3503872.

[5] Corradini, I. (2020). *Building a cybersecurity culture in organizations*. Cham: Springer.

[6] Dawes, J.H., Shin, D., & Bianculli, D. (2023). Towards log slicing. In *International conference on fundamental approaches to software engineering* (pp. 249-259). Cham: Springer.

[7] Digital 2023: Global overview report. (2023). Retrieved from https://datareportal.com/reports/digital-2023-global-overview-report.

[8] Gerhards, R. (2009). *RFC 5424 – the syslog protocol*. Fremont: Internet Engineering Task Force. doi: 10.17487/RFC5424.

[9] Joyce, C., Roman, F.L., Miller, B., Jeffries, J., & Miller, R.C. (2021). Emerging cybersecurity threats in radiation oncology. *Advances in Radiation Oncology*, 6(6), article number 100796. doi: 10.1016/j.adro.2021.100796.

[10] Kabamba, H.M., Khouzam, M., & Dagenais, M. (2023). Advanced strategies for precise and transparent debugging of performance issues in in-memory data store-based microservices. *arXiv – Computer Science*, article number arXiv:2311.11230. doi: 10.48550/arXiv.2311.11230.

[11] Karpowicz, M.P. (2021). Covid-19 pandemic and internet traffic in Poland: Evidence from selected regional networks. *Journal of Telecommunications and Information Technology*, 3, 86-91. doi: 10.26636/jtit.2021.154721.

[12] Khan, N.A., Brohi, S.N., & Zaman, N. (2020). Ten deadly cyber security threats amid Covid-19 pandemic. *TechRxiv*, 1-7. doi: 10.36227/techrxiv.12278792.v1.

[13] Landauer, M., Wurzenberger, M., Skopik, F., Hotwagner, W., & Höld, G. (2023). AMiner: A modular log data analysis pipeline for anomaly-based intrusion detection. *Digital Threats: Research and Practice*, 4(1), 1-16. doi: 10.1145/3567675.

[14] Li, H., Shang, W., Adams, B., Sayagh, M., & Hassan, A.E. (2021). A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering*, 47(12), 2858-2873. doi: 10.1109/TSE.2020.2970422.

[15] Liang, F., Hatcher, W., Liao, W., Gao, W., & Yu, W. (2019). Machine learning for security and the internet of things: The good, the bad, and the ugly. *IEEE Access*, 7, 158126-158147. doi: 10.1109/ACCESS.2019.2948912.

[16] Ma, H., Pljonkin, A., & Singh, P.K. (2022). Design and implementation of Internet-of-Things software monitoring and early warning system based on nonlinear technology. *Nonlinear Engineering*, 11(1), 355-363. doi: 10.1515/nleng-2022-0036.

[17] Marin-Castro, H.M., & Tello-Leal, E. (2021). Event log preprocessing for process mining: A review. *Applied Sciences*, 11(22), article number 10556. doi: 10.3390/app112210556.

[18] Observability vs. monitoring: What's the difference? (2022). Retrieved from https://www.ibm.com/blog/observability-vs-monitoring/.

[19] Shaukat, K., Luo, S., Varadharajan, V., Hameed, I.A., & Xu, M. (2020). A survey on machine learning techniques for cyber security in the last decade. *IEEE Access*, 8, 222310-222354. doi: 10.1109/ACCESS.2020.3041951.

[20] Sombatruang, N., Omiya, T., Miyamoto, D., Sasse, M.A., Kadobayashi, Y., & Baddeley, M. (2020). Attributes affecting user decision to adopt a virtual private network (VPN) app. In W. Meng, D. Gollmann, C.D. Jensen & J. Zhou (Eds.), *Lecture notes in computer science* (pp. 223-242). Cham: Springer. doi: 10.1007/978-3-030-61078-4_13.

[21] Torres Martínez, J., Iglesias Comesaña, C., & García-Nieto, P.J. (2019). Review: Machine learning techniques applied to cybersecurity. *International Journal of Machine Learning and Cybernetics*, 10, 2823-2836. doi: 10.1007/s13042-018-00906-1.

[22] Trautsch, F., Herbold, S., & Grabowski, J. (2020). Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects. *Journal of Systems and Software*, 159, article number 110421. doi: 10.1016/j.jss.2019.110421.

[23] Wang, J., Liu, B.J., He, W., Xue, J.K., & Han, X.Y. (2021). Research on computer application software monitoring data processing technology based on NLP. *IOP Conference Series: Materials Science and Engineering*, 1043, article number 032021. doi: 10.1088/1757-899X/1043/3/032021.
[24] Winston – a logger for just about everything. (2024). Retrieved from https://www.npmjs.com/package/winston.
[25] Yadav, R. (2021). Cyber security threats during Covid-19 pandemic. *International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies*, 12(3), 1-7. doi: 10.14456/ITJEMAST.2021.59.
[26] Zhylin, A.V., Shapoval, O.M., & Uspensky, O.A. (2020). *Information security technologies in information and telecommunication systems*. Kyiv: Polytechnica.

Про особливості впровадження методу адаптивного логування

Ілля Супруненко
Аспірант
Черкаський державний технологічний університет
18006, бульв. Шевченка, 460, м. Черкаси, Україна
https://orcid.org/0000-0002-1188-4804

Володимир Рудницький

Доктор технічних наук, професор, головний науковий співробітник
Черкаський державний технологічний університет
18006, бульв. Шевченка, 460, м. Черкаси, Україна
Державний науково-дослідний інститут випробувань і сертифікації озброєння та військової техніки
18000, вул. В'ячеслава Чорновола, 164, м. Черкаси, Україна
https://orcid.org/0000-0003-3473-7433

Анотація. Актуальність цієї роботи полягає в тому, що розуміння причин чому певний код виконується певним чином, як протягом нормального циклу виконання, так і у випадку помилок, є важливою частиною хорошого дизайну програмного забезпечення. Оскільки комп'ютерні системи стають складнішими, запит на рішення, що дозволяють глибше поглянути в процес виконання коду, лишається на високому рівні. Метою даної роботи є формалізація програмного інструменту, здатного забезпечити кращу спостережність програми. Основними методами роботи є: аналіз поширених підходів, таких як моніторинг та логування, формалізація основних компонентів та моделювання прикладу реалізації на основі патерну програмування Singleton. В результаті було проаналізовано логгінг на основі «тільки критичність» і аналогічним чином описано основні частини «методу адаптивного логування». Є дві відмінні риси цього методу: тегування лог повідомлень і подальше введення схеми конфігурації, яка здатна адаптуватися до мінливих вимог під час виконання програми. Системи, що використовують такий підхід, мають можливість отримувати більш точну інформацію про хід виконання, а також можуть зосередитися на певних компонентах, які можуть поводитися некоректно. Оскільки таке перемикання відбувається без перезапуска програми, за якою спостерігають, має бути можливість налагоджувати та досліджувати деякі проблеми без необхідності намагатися відтворити з нуля стан середовища, в якому вони виникли. Також представлено приклад формального опису на основі патерну програмування Singleton, який описує методи та їхні сигнатури, необхідні для створення базового варіанту адаптивного методу логування. Цей підхід може бути використаний різними програмами та мовами програмування, оскільки він розроблений в загальних рисах і всі необхідні абстракції повинні бути присутніми в різних середовищах

Ключові слова: кібератаки; безпека інформації; спостережність; дебагінг; адаптивний підхід; прикладна модель