Complex Analysis of the Behavior of Applications Running in the Critical Infrastructure Environment

Deniss-Bogdan ONOFREI-RIZA
National Institute for Research & Development in Informatics - ICI Bucharest
deniss.onofrei@ici.ro

Abstract: Everybody knows that famous quote “to err is human”. Therefore, it is impossible for a human being to create something flawless, especially from a single movement reflected in the effort of software/hardware developers. The only way human beings can truly achieve perfection is to try again and again. The concept of testing (or the attack supported by the need to discover vulnerabilities) is something similar; it leads to the idea of trying to continually test something that has been built/developed, in order to ensure that it works as intended (perfectly). Like any other thing that has been generated in this era of the Internet, software is an extremely complicated creation, with thousands of modular parts. Although these sequences are not mechanical (like into a machine), but transposed into code lines, they are equally sensitive if not even more. So, the chance for one of these to malfunction is extremely high (even one line of code generating “turbulence” is so strong that it could lead to the destruction of the entire software).
Keywords: Critical infrastructures, secure policy, software testing, software development, software engineering, testing tools, advanced metrics, data flow, software integration, data security, security risks, testing frameworks, best practices.

INTRODUCTION

This study brings to the fore the complete ways of testing software architectures treating this case as a rather important task which needs to be performed in several stages and which involves the existence of specialists prepared for operations of this type. By default, all of these things indicate that testing processes have an increased cost (due to the high number of hours of work spent to deeply examine the desired products).

This article also responds somewhat to the question of why companies are willing to invest so much time and money on testing their own software (and not only).

This is why programmers and software developers rely on testing to ensure that no error or mistake that has been made in the code remains unverified. They create sophisticated testing mechanisms and develop their software to work in difficult and strange situations, therefore complying with market requirements.

https://doi.org/10.54851/v4i2y202206

WEB AND DESKTOP APPLICATIONS

Unlike modern attacks, those used in the past speculated on vulnerabilities discovered in the implementation of the network stack. Most often than not, they are conducted through the social engineering method. Then, the target migrates to applications (as they are a common place where users interact with other entities). Therefore, there are several attacks on web applications (these are highlighted as a starting route to the desktop applications). This type of attack allows examiners/testers to gain access to data or other systems on the network. XML External Entity Processing allows the user of XML files to gain access to functions. The SQL injection attacks not only can allow attackers to gain access to data stored in the database, but, in some cases, they can also be used to gain access to the operating system.

Not all attacks are about server infrastructure. A cross-site scripting attack aims to gain access to data held by a user. In addition to data related to the operating system, this may also include the data stored on websites to which the user has access. Session identification information (stored in cookies) may be used to gain access to other systems where the user has privileges.

Web applications can be protected in several ways/technics when they are developed. Firstly, all data entries should be validated by each function, even if they are generated from reliable sources. In addition, nothing should be sent directly from the user to a subsystem (such actions facilitating the "command injection" attacks).

When it comes to Application layer exploitation, attackers are looking for ways to insert their own code in the memory space of the application. The goal is to control the flow of the application, altering the place where the processor gets its instructions to execute. The buffer overflow attacks can be used to „push" instructions and return addresses to stack (where the application data is stored). If an attacker can do this, the program can be manipulated to execute those instructions (ACE). Another type of attack involves the structure of memory called heap, where dynamic data is stored. The heap spraying attack implies the insertion of the code from the attacker into the heap. Once there, the attacker can make the program execute the instructions he wants.

Once the attacker has presence and persistence in the targeted environment, he seeks ways to propagate laterally to gain access to other systems. This may include escalation of privileges and studying to gain access to systems containing more data.

DEFENSE AND RESOURCE PROTECTION TECHNIQUES

The use of a (depth) defense approach to network protection is a long-developed and respected strategy. It is focused on developing several defense thresholds (to make it more difficult for attacks to gain access to the network or traffic data). This ignores modern opponents who are organised and well-funded. If there are no dedicated detecting strategies and controls, the slowdown of the attack will only give a short delay in speculating about the security breach. Defence in breadth is a way to alleviate some of the deep defense concerns by widening the scope of understanding how modern attackers work. A defence in breadth approach targets the entire network stack, providing controls, where it is possible, to protect against attacks.

There is a new approach in the network architecture, called the "defensive network architecture". It is based on the idea to lead social engineering attacks towards usual stuff. It also takes into account the importance of visibility and response to the incident (because security breaches are common and cannot be always avoided). Memorising network and/or system events may be essential to detection (when many logs are collected, a system that can manage them is needed- SIEM).

Encryption is an important concept, because confidentiality is very important (especially when attackers are looking for any advantage they can get). If they can intercept messages that are not encrypted, they may use the content of the message by speculation. In parallel, users will sometimes make the mistake of believing

that messages sent to other users within a system are safe (because they remain inside). By speculative intercepting and use, unencrypted messages may generate vulnerable events, even at the level of encryption of memory/ storage devices.

It cannot be assumed that a disc that has been encrypted is safe. Once someone has logged in as a legitimate user, the disk is unencrypted. This means that if an attacker can obtain authenticated access, even by introducing malware that is run as a main user, the stored information may be accessed.

When taking into account the final result, there are two types of encryptions. The first is a substitution, where one character is replaced by another. This is common in encryption schemes such as rotation cipher and/or Vigenere cipher. The second type consists in a transformation mechanism (the unencrypted message, or the simple text, is not replaced at the character level, but the whole message is generally transformed by a mathematical process). This transformation can be achieved with fixed lengths of the message representing a block cipher; it can also be applied byte by byte – how a flow cipher works. With a block cipher, the data size is expected to be a multiple of the block size (the final block requiring reinterpretation to reach the correct size).

When talking about encryption, key management is essential. An important element of it may be the key generation. Pre-shared keys may be used, which may be learned or intercepted at the time of sharing. If a pre-shared key is not used, the key could be derived. This can be done by employing the Diffie-Hellman Key Exchange protocols. Using a common starting point, both sides of the process add value and transmit it to the other party. Once the value has been added to the shared key, both communicating parties use the common value (plus random value from A plus random value from B); after negotiating the common key, communicating entities may start sending encrypted messages. Processes of this type could be used for symmetrical keys, where the same key is used for both encryption and decryption. The advanced encryption standard (AES) is a common symmetric key encryption algorithm. The AES supports 128, 192 and 256 bits. An asymmetric key algorithm may also be used, in which different keys are employed for encryption and decryption. Sometimes, this is called “public key encryption”. A common approach to this is the use of a hybrid crypto-system in which public key cryptography is used to share a session key (a symmetric key employed to encrypt messages within the session).

The certificates, defined by X.509, which is a subset of the X.500 digital directory standard, are used to store information about the public key. These include data related to the identity of the certificate holder so that certificate verification can be made.

Certificates may be managed using a CA (the trusted side that verifies the identity of the certificate holder). However, a CA is not the only way to verify the identity. PGP uses the trusted web model, in which individual users validate identity by signing the public key to the people they “know”.

A MAC is used to ensure that messages have not been changed. This is generally a cryptographic hash-algorithm that generates a fixed length assimilation value from variable length data; it can be used not only for message authentication, but also for checking whether the files have been modified/corrupted or not.

Full disc encryption is a commonly integrated technique in current operating systems. Windows developments usually use BitLocker, although there are third-party software solutions that can perform somewhat the same functions; FileVault can be identified on macOS, while Linux uses a dm-crypt incorporated into the kernel (which requires software to be installed to manage volume encrypting and use). As with any encryption, the key management is essential, and encrypted files or systems are not protected against any forced access action. Encrypted files are exposed to data loss (implicitly, the key or the encryption password).

DATA CLASSIFICATION

Another element that is worth considering is the classification of data. As the main activity, this operation helps to identify all data resources, as well as prioritise their sensitivity or importance. This action is vital in an attempt to implement a security model. For example, the Biba cannot be implemented if sensitivity or priority is not known (since this technique needs to know who can access reading a file at the top level and who can write down). The Biba security model refers to data integrity. The same principle applies to the Clark-Wilson integrity model. Other models, such as Bell-LaPadula, do not value integrity that much, but privacy. On the one hand, integrity ensures that data is not altered or corrupted by unauthorised users. On the other hand, privacy ensures that the data is not viewed by those who do not have access. Security models are required to implement mandatory access controls.

SOFTWARE ARCHITECTURE MODELS

The software architecture model plays an important role in the ability of the final product to be scalable and to meet the requirements of end-users as appropriate. For this reason, the behaviour of applications based on the main software architecture models will be studied in this paper (trying to discover how software components run and interact).

By minimising the risk of commercial propagation, the limitations of performance are also minimised, as such architectural models are nothing but reusable structural schemes that may contain predefined sets of subsystems (but also roles, responsibilities, rules and/or traffic related to them).

Therefore, the following types will be studied (see Figures 1-9):

a) layer-based software architecture model;
b) event-based architecture model;
c) kernel-based architecture model;
d) microservices-based architecture model;
e) shared memory-based architecture model;
f) client – server architecture model;
g) master – slave architecture model;
h) data flow filter architecture model;
i) peer-to-peer (P2P) architecture model.

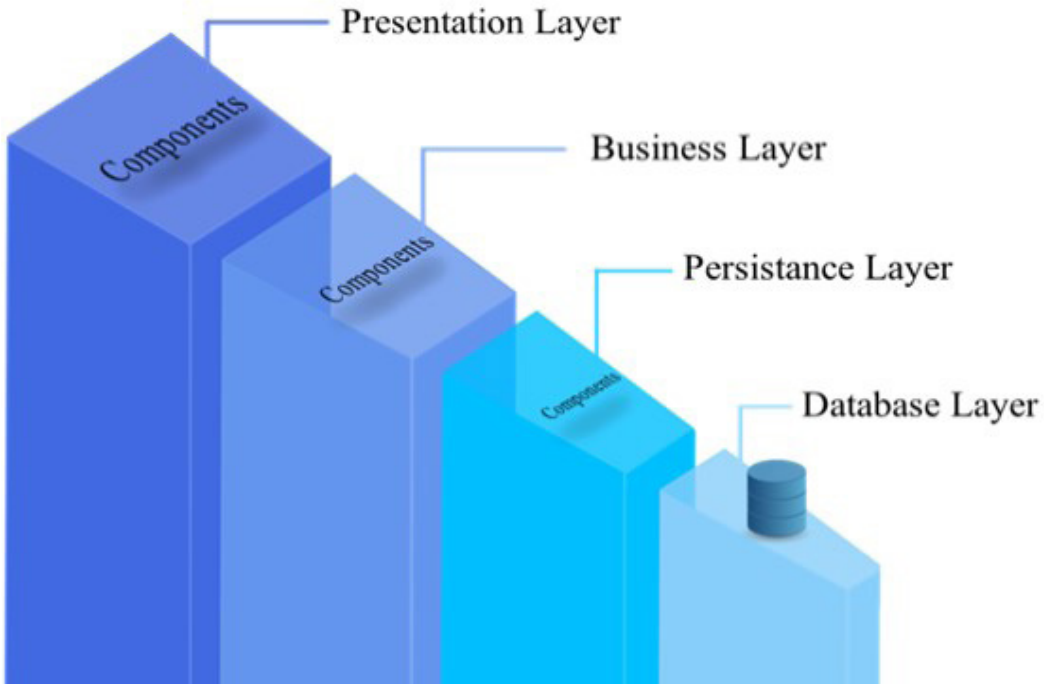

Fig. 1: *Layer-based software architecture model*

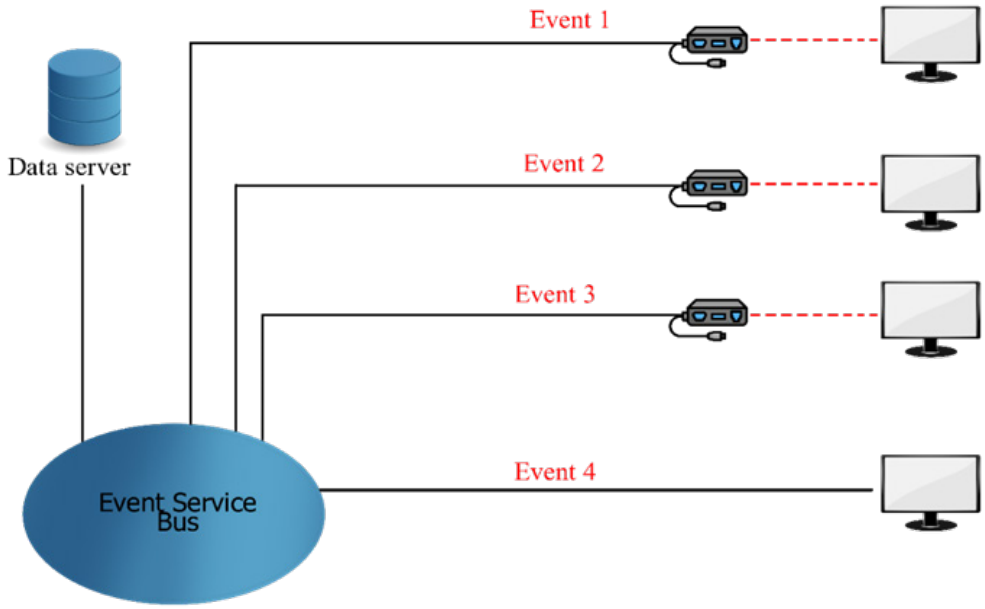

Fig. 2: *Event-based architecture model*

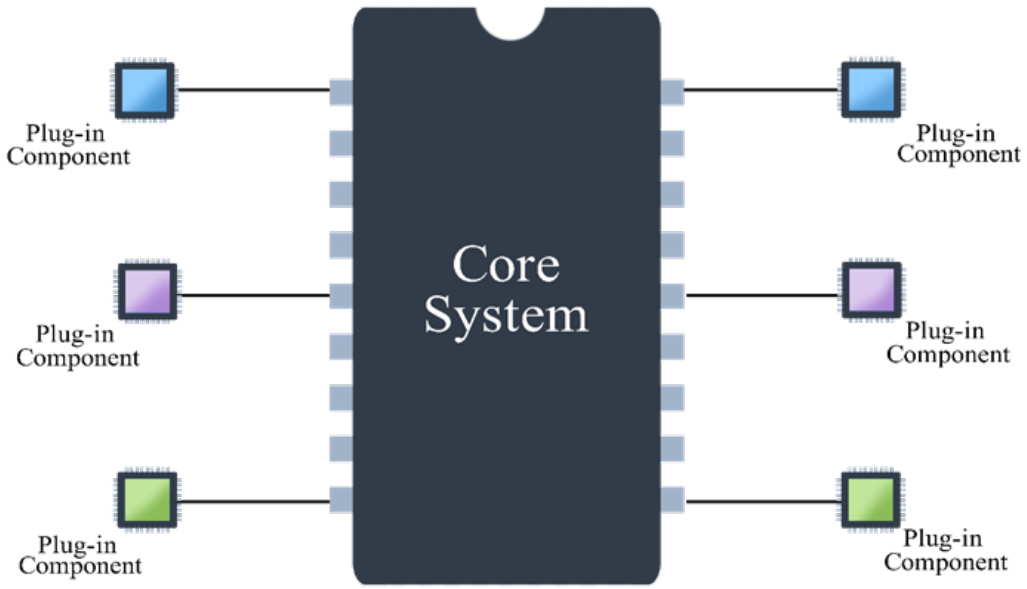

Fig. 3: *Kernel-based architecture model*

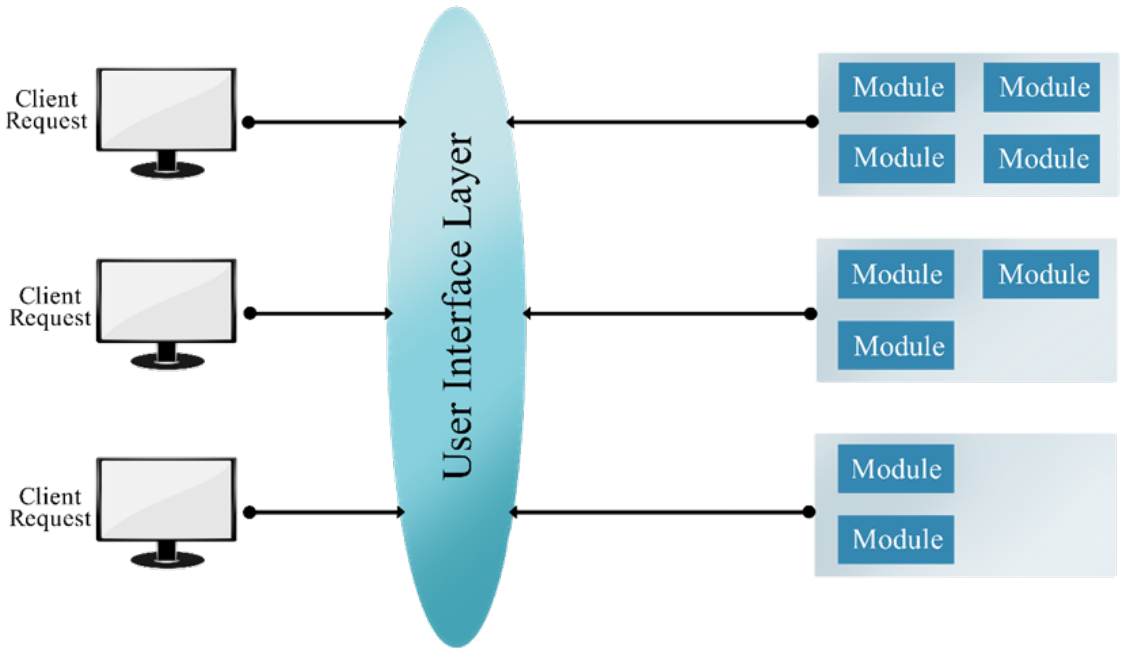

Fig. 4: *Microservices-based architecture model*

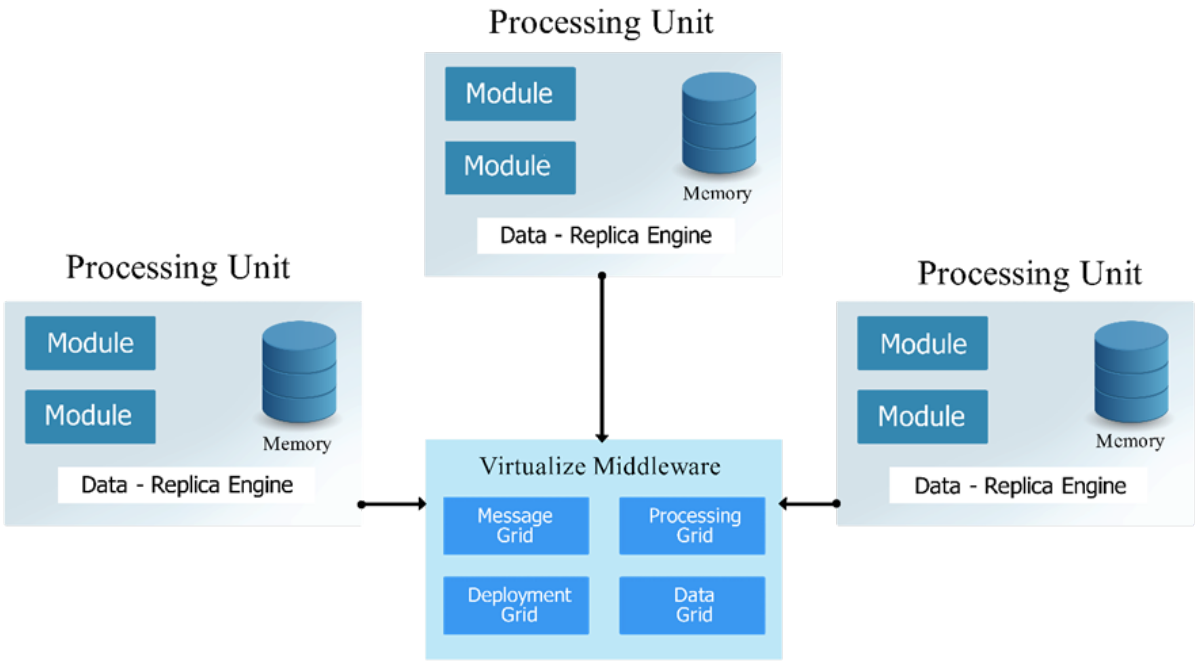

Fig. 5: *Shared memory-based architecture model*

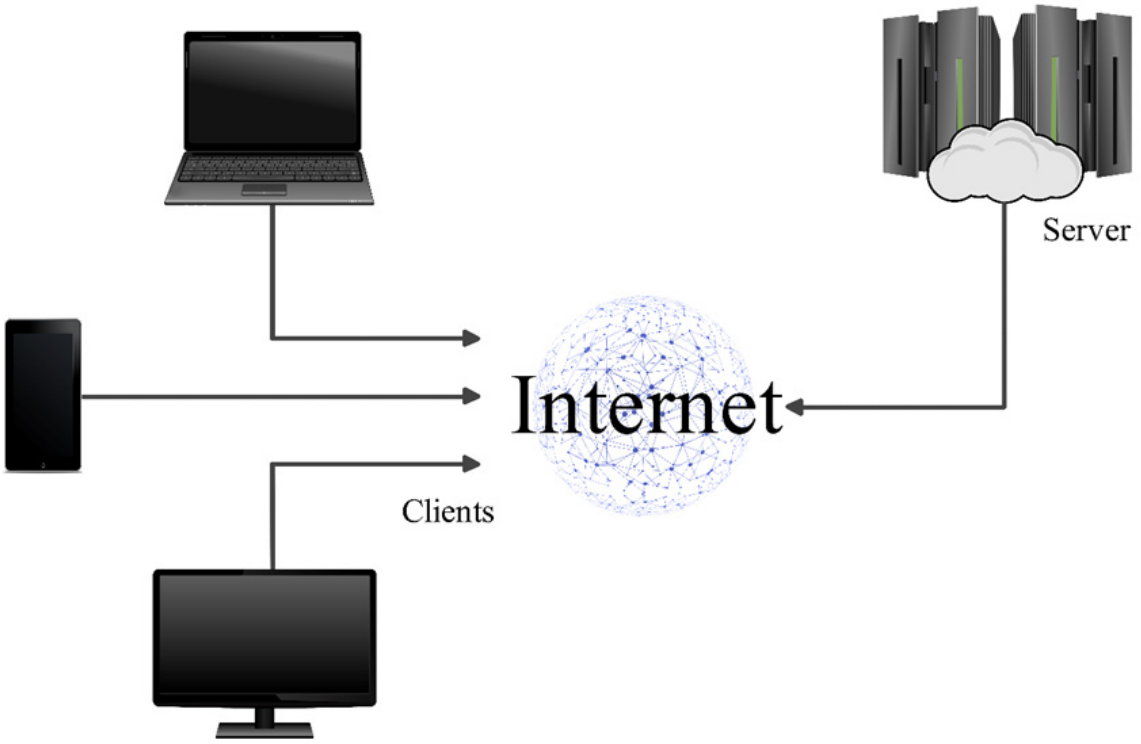

Fig. 6: *Client – server architecture model*

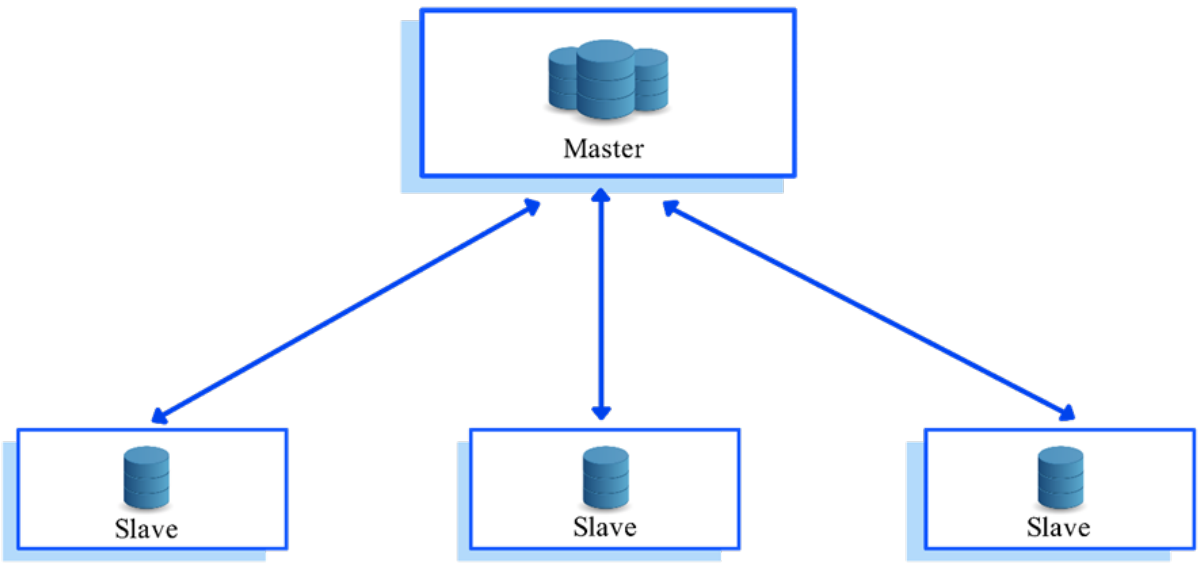

Fig. 7: *Master – slave architecture model*

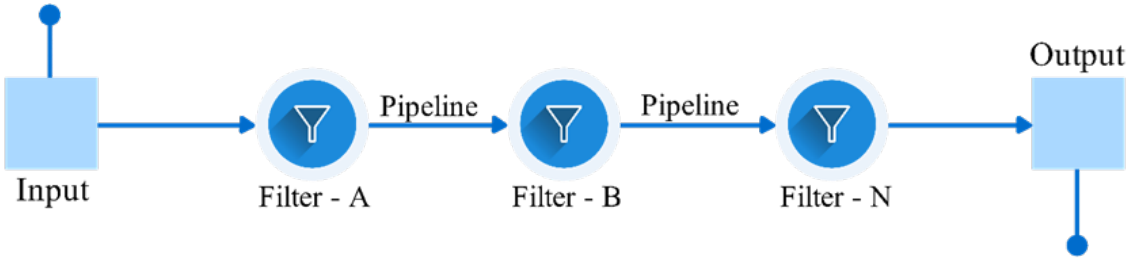

Fig. 8: *Data flow filter architecture mode*

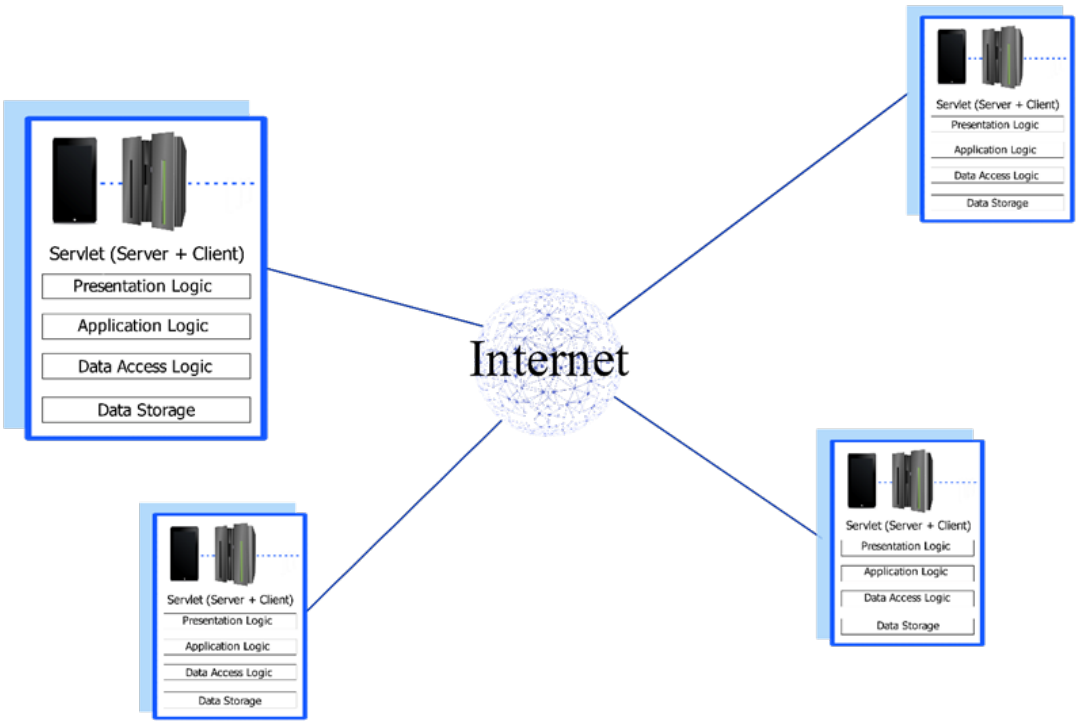

Fig. 9: *Peer-to-peer (P2P) architecture model*

APPLICATION ARCHITECTURE

At the application design level, it is recommended to use known architectures or frequently used design patterns (with results). Native self-contained applications do not have an external architecture, but may have an internal one. Thus, being self-reliant, they are not based on other systems or services. A common application architecture is the n-tier or multi-tier type. The multi-tier architecture comprises the following levels: Presentation, Application, Business Logic and Data Access. Sometimes, the Application and Business Logic layers are consolidated precisely to manage the logic of the application based on business requirements. This would be a three-level architecture (three-tier), an implementation of the MVC application design.

Web-type applications generally use a multi-tier architecture. Thus, the Browser represents

the Presentation layer (view). There is an application server running, namely Java, .NET, PHP or other programming language that manages the logic of business and application (controller). Finally, there is also the data storage, most likely in the form of a database, which is the Data Access layer.

Modern applications still use multi-tier architectures, which are often divided into functions or services. When an application is viewed or designed in this way, it has a service-oriented architecture (SOA), which means that it is divided into services that interact with each other. This architecture provides a modularity so that any service can be replaced by another (with the same input/output specification), without the rest of the application being altered. Lately, this approach has been adapted to generate micro-service architectures. Micro-services are activated via containers such as Docker or Kubernetes. Sometimes, these containers are implemented through a cloud service provider. Traditional application architectures may also be implemented using such a supplier, generating a hybrid result – some parts of the application are local, while others are implemented through a cloud service provider. These suppliers are beginning to expose application developers to new design modes. Thus, functions that are connected to create an application have been developed, but there is no server behind them to which the attacker can gain access. Similarly, the use of containers led to infrastructure automation (so that containers and virtual machines are built and destroyed upon request). An attacker who obtained access to such an environment must try again, when the system he gained access to has disappeared.

Applications often need to store data, whether temporary or persistent. Traditionally, the application data was stored in relational databases accessed using languages such as SQL. Modern applications evolve from this approach and start using NoSQL databases (which may use semi-structured documents or key values associative matrices).

CONCLUSIONS

Static techniques include both testing and static analysis, thus representing effective methods to identify and eliminate defects (supplementing the results of dynamic testing).

The testing is defined throughout the entire life cycle, both static and dynamic, and refers to the planning, preparation and evaluation of software and related products. The two approaches can be used to achieve these goals. Static analysis is an automated form of static testing.

The definitions of static analysis and static testing are very similar, which may lead to confusion. The definition of static analysis applies to the forms of static testing that are not part of static analysis. In general, static analysis involves the use of specific instruments. The most important difference is: given the source code to be evaluated, dynamic testing effectively performs the code, while static testing (including static analysis) does not perform these sequences subject to evaluation.

In the case of analysis processes, there is no correct way considered to carry them out. Instead, there are a lot of ways in which such a process can fail, due to several factors (ranging from organizational to human factors).

When talking about the success of an analysis process, one of the most important aspects is the presence of a leader in the project. He/she must be a person with expertise, enthusiasm and practical thinking, in order to be able to guide the other team members.

The tools used in the analysis process must be updated (and in good working conditions) to facilitate the achievement of appropriate results. In addition to all these instruments, the checklists should also be constantly updated. If a defective product is identified, it is recommended to introduce a record in the future, that will facilitate the discovery of similar types of defects (faster).

At regular intervals, it is also recommended to take a full analysis of the checklist, in order to remove elements that are no longer relevant to the current testing.

Organisations should consider a security architecture, even unrelated to the application or network design. Thus, there is a set of data and methodologies that guide the implementation of security within the organisation. NIST (National Institute of Standards and Technology – elaborator of security measurement standards and/or its implementation) recommends the five functions – Identification, Protection, Detection, Response and Recovery, both organizational and personnel – and it also recommends how information security or any other potential risks should be assessed.

NIST is not the only organisation with security recommendations, as ISO 27001 is doing the same thing. The latter recommends Planning, Execution, Verification and Acting. At the same time, there is a life cycle of the attack that identifies the phases by which an opponent attempts to gain access to critical systems or data. These are initial recognition, initial compromise, establishment of support, escalation of privileges, internal recognition, side movement, maintenance of persistence and mission accomplishment.

BIBLIOGRAFY

Ball, T., & Eick, S. G. (1994). *Visualizing Program Slices*. [Conference Presentation]. IEEE Symposium on Visual Languages, California.

Beck, K. (1994). *Simple smalltalk testing: with patterns.* (The Smalltalk Report 4.2), 16-18. Retrieved from http://www.xprogramming.com/testfram.htm

Binder, R. V. (1996). Testing object-oriented software: a survey. *Software Testing Verification and Reliability*, *6*(34), 125-252. DOI: 10.1002/(SICI)1099-1689(199609/12)6:3/43.0.CO;2-X

Black, R. (Ed.). (2009). *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing* (3rd ed.). New York, NY: John Wiley & Sons.

Buwalda, H., Janssen, D., & Pinkster, I. (2001). *Integrated Test Design and Automation: Using the Testframe Method.* Boston: Addison Wesley.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. *IEEE Transactions of Software Engineering*, *20*(6), 476-493. DOI: 10.1109/32.295895

Clarke, L. A. (1989). A formal evaluation of dataflow path selection criteria. *IEEE Transactions on Software Engineering, 15*(11), 1318-1332.

Crowdbotics. (n.d.). *5 Common Software Architecture Patterns and When to Use Them*. https://www.crowdbotics.com/blog/5-common-software-architecture-patterns-and-when-to-use-them

Dang, A. T. (2020, October 5). *Software Architecture: The Most Important Architectural Patterns You Need to Know. Level Up Coding – gitconnected.* https://levelup.gitconnected.com/software-architecture-the-important-architectural-patterns-you-need-to-know-a1f5ea7e4e3d

DataDome. (2019). *Web application security best practices*. https://datadome.co/bot-management-protection/web-application-security-best-practices/

Gaffney, C., Trefftz, C., & Jorgensen, P. (2004). Tools for coverage testing: necessary but not sufficient. *Journal of Computing Sciences in Colleges*, *20*(1), 27-33.

Gallagher, K. B., & Lyle, J. R. (1991). Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, *17*(8), 751-761.

GitHub. (2018). *Pragmatists / JUnitParamsPublic.* https://github.com/Pragmatists/JUnitParams

Gregory, J., & Crispin, L. (Eds.). (2015). *More Agile Testing: Learning Journeys for the Whole Team*. Indiana: Addison-Wesley.

Hoffner, T. (1995). *Evaluation and comparison of Program Slicing Tools* (Technical Report LiTH-IDA-R-95-01). Department of Computer and Information Science, Linkoping University, Sweden. https://www.ida.liu.se/publications/techrep/95/trl95.html

ISO. (2011). *ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models* (Technical Report). International Organization for Standardization https://www.iso.org/standard/35733.html

ISO. (2015). *ISO/IEC/IEEE 29119-4:2015. Software and systems engineering – Software testing – Part 4: Test techniques* (Technical Report). International Organization for Standardization https://www.iso.org/standard/60245.html

Jorgensen, P. C., & Erickson, C. (1994). Object-oriented integration testing. *Communications of the ACM*, *37*(9), 30-38.

Kramer, A., & Legeard, B. (2016). *Model-Based Testing Essentials: Guide to the ISTOB Certified Model-Based Tester: Foundation Level.* New York, NY: Wiley.

Mokito. (n.d.). T*asty mocking framework for unit tests in Java.* https://site.mockito.org/

NI Business Info. (n.d.). *Benefits of databases. Types of database system.* https://www.nibusinessinfo.co.uk/content/types-database-system

OWASP. (2021). *OWASP Top Ten Web Application Security Risks.* https://owasp.org/www-project-top-ten/

Rapps, S., & Weyuker, E. J. (1985). Selecting software test data using dataflow information. *IEEE Transactions on Software Engineering*, *11*(4), 367-375.

Wikipedia. (n.d.). *Application security*. https://en.wikipedia.org/wiki/Application_security

Wikipedia. (n.d.). *List of unit testing frameworks.* https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

Wikipedia. (n.d.). *Unit testing.* https://en.wikipedia.org/wiki/Unit_testing