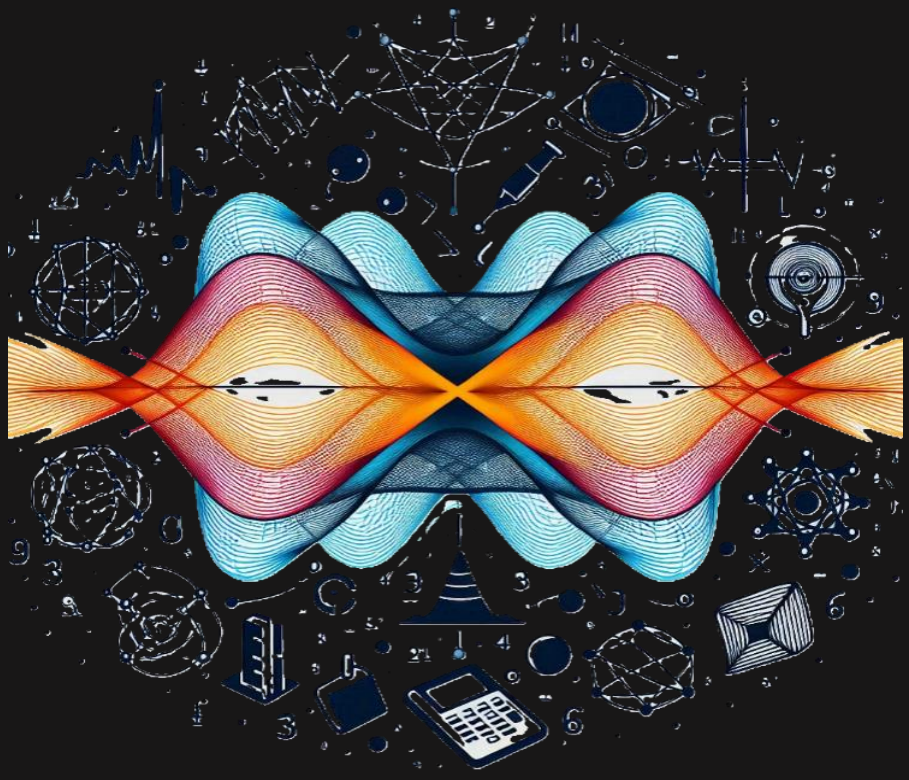

# Systemology and Matrix of Functions and Systems

A journey from system thinking to mathematical models to understand software systems analysis and synthesis.

Written by Milad Kahsari Alhadi

Ai000 Cybernetics QLab
«the ark in the digital world»

# Table of Contents

# Research Document Version

In the table 1, the changes that each person has applied to this document are listed separately according to the date and titles added, deleted or modified.

| Date | Description |
| --- | --- |
| 2022-May-14 | The first version of this research paper has been written by Milad Kahsari Alhadi. In the first version of this paper, I try to provide a standard model to think, to understand and also to synthesis systems to their base components. |
| 2024-May-05 | The second edition received changes and updates in the basic concepts of systems thinking, systems modeling, and also the resolution of some ambiguities in the provided concepts and photos. |
| 2024-August-20 | After reviewing this book, Dmitry written a foreword. This foreword completely shows the importance of systems engineering concepts for every computer science researcher. |

**Table 1: Document Changes and Modification Records**

# Foreword by Dmitry Vostokov

In an increasingly interconnected world, where the complexity of our systems grows exponentially, understanding the intricate fabric of relationships that underlies our technological, social, and economic landscapes has never been more critical. The Systemology and Matrix of Functions and Systems is a timely and essential introductory resource for anyone seeking to navigate the challenges posed by modern security environments.

The concept of security has evolved beyond simple safeguards and protective measures. In our digital age, security must be understood as an integral part of a larger, dynamic system—a system that includes not only the technologies we rely on but also the human, organizational, and environmental factors that shape our interactions with these technologies. To truly grasp the nature of threats and vulnerabilities, one must adopt a systems-thinking approach that views security not as an isolated issue but as a holistic concern that runs through every layer of our complex global infrastructure.

The Systemology and Matrix of Functions and Systems fills an important gap in cybersecurity literature. It is structured into two complementary parts. The first part introduces the foundational concepts of general systems theory and system thinking. Here, you will understand the general principles that govern all systems—natural or artificial—and how these principles can be applied to analyze and predict system behavior. It outlines the characteristics of systems, such as interdependence, feedback loops, synergy, emergent properties, and classification of systems, providing readers with the tools to think critically about how systems function and interact. This part provides many examples of systems, including those about computer hardware and software.

The second part builds upon this foundation, delving into the specific application of systems theory to security and threats with additional examples. It also refines basic system security vocabulary.

As you start your systems thinking intellectual journey, you will be challenged to rethink traditional approaches to security and encouraged to consider the broader context in which threats emerge and recognize the interconnectedness of various system components in an ever-changing threat landscape.

The Systemology and Matrix of Functions and Systems is a much-needed introductory guide to understanding the complex systems that define our world. Whether you are a security professional, a software or systems engineer, or simply someone interested in the dynamics of complex systems, it will provide you with the knowledge and insights needed to approach security from a holistic, systems-oriented perspective. In a time when security threats are becoming increasingly sophisticated and diverse, the systems approach outlined in the Systemology and Matrix of Functions and Systems offers a powerful framework for developing the strategies and solutions necessary to protect our most vital systems.

**Dmitry Vostokov / Software Diagnostics Institute.**

# Introduction

Ever since I was a young boy, I've been captivated by the phenomena around me. I've always sought to understand everything deeply, searching for views and theories that can elucidate complex concepts in the simplest possible terms. I inherently sensed connections between things but struggled to grasp their essence and dynamics. My introduction to systems thinking is a blur, but its impact on my life has been profound. Through books, articles, and insights from eminent engineers and scientists, I've gained unwavering confidence in this approach. It has empowered me to swiftly comprehend diverse subjects—from computer science to intricate fields like aerodynamic design. A system isn't just any collection of things. A system is an interconnected set of elements/objects or things that is coherently organized in a way that achieves something or generate an output. As you can see from this description of systems, a system must consist of three kinds of things: elements, interconnections, and a function or purpose. Understanding this interconnection, functions and purpose is a key to understand every phenomenon in our environment.

In this book, I aim to share my perspective on systems thinking and its application in solving complex problems, particularly in computer science and cybernetics systems security. I'll delve from the highest conceptual levels to intricate details, focusing on next-generation cybernetic systems integrating AI and addressing vulnerabilities and threats in everyday life and industry. Welcome to this exhilarating journey! Expect to encounter fascinating insights along the way. Keep reading until the end and share your thoughts with me via email once you've applied these concepts in your career or daily life. I'm eager to hear your feedback—it fuels my passion for this work. However, folks, remember this in our first steps to reading of this book: Systems permeate everything; understanding them offers a glimpse into the creator's mind. System theory truly encompasses the theory of everything. With this knowledge, you can understand, comprehend, explain and also solve every problem. Believe me, system thinking with critical thinking is the only way to autonomy. The time will proof it.

# What is a System?

There are numerous definitions of system and systems thinking, reflecting the diversity of understanding across disciplines. While the concept of a system may seem intuitive, there is no universally accepted definition among researchers and practitioners of systems thinking. The concept was first articulated by Physician and Philosopher Alexander Bogdanov and biologist Ludwig von Bertalanffy in the 1950s through his proposal of the General Systems Theory, which sought to integrate various disciplines by focusing on the similarities within complex systems. This theory has since fostered significant contributions from professionals in fields such as management, engineering, sociology, economics, and environmental studies, enhancing the

application and development of systems thinking. Systems thinking is fundamentally concerned with comprehending observable entities within our universe. It adopts an analytical and philosophical approach that emphasizes the entirety, complexity, and interconnectivity of physical or metaphysical phenomena. Unlike traditional analysis, which may focus on individual components, systems thinking prioritizes understanding the interactions and influences between components, embodying the principle that "more than the sum of its parts." This approach is applied across various domains, including management and engineering, and even in understanding human anatomy and behavior, considering humans as complex, multidimensional systems.

Philosophically, it is suggested that all observable phenomena in the universe can be described as systems. This includes not only tangible phenomena but also intangible ones, such as Earth's gravity, the movements of planets, or the functionality of the Internet. Thus, defining systems solely in material terms is restrictive and does not accommodate the broader spectrum of phenomena that can be systemically analyzed. We inhabit a world comprised of both physical and metaphysical systems, and a deep understanding of their nature can offer insights into solving complex problems. In essence, a system is an organized set of components or subsystems interacting to achieve a specific objective. Systems process various inputs through specific mechanisms to produce outputs, analogous to functions in mathematics or structured programming languages like C. Just as a mathematical function or a programming function processes inputs to generate an output, systems employ similar mechanisms to fulfill their objectives. From this abstract viewpoint, systems parallel the functionalities observed in both mathematical and programming contexts, demonstrating their foundational role in understanding and modeling our world.

**Consider the straightforward function depicted in Figure 1, which takes three inputs represented by the symbols X, Y, and Z. These inputs are processed using the formula ((X + Y + Z) * 2) within the specified function. The result of this computation is then output as the symbol R. This process is illustrated using mathematical notation in Figure 1, effectively demonstrating the function's operational mechanics.**

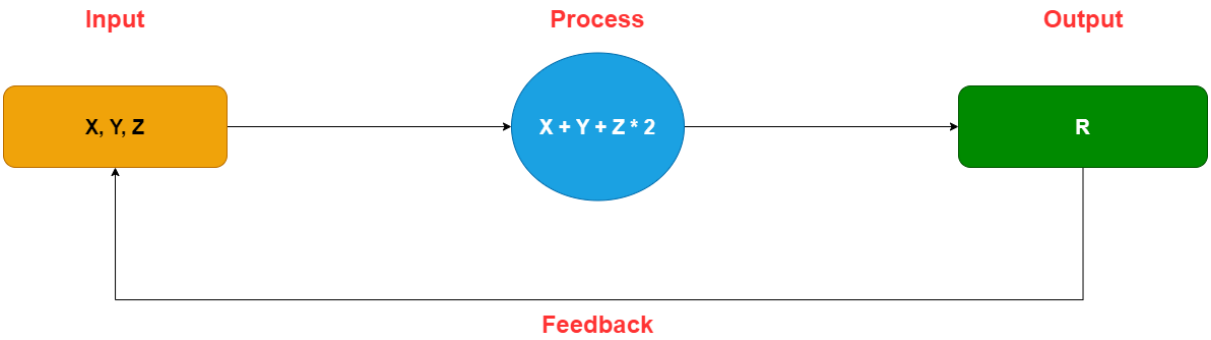


Figure 1: Representation of a System in the form of Mathematical Equations

In Figure 2, a specific function is delineated and implemented in the C programming language. This function accepts three parameters, processes these inputs through a series of calculations, and ultimately yields an output. It is pertinent to note that our world can be analogously understood as comprising functions or, more abstractly, as a system. By defining functions and establishing correct connections between them, any phenomenon can be modeled, analyzed, and even replicated. On a lighter note, given this perspective, one might jest that the introduction of the object-oriented paradigm in software development was superfluous. This paradigm, while innovative, arguably introduced additional complexity into the software industry. This remark is made in jest, as the author humorously reflects on the intricacies of software paradigms.

```cpp
#include "headers/require.h"

int SystemX(int arg_x, int arg_y, int arg_z)
{
    return (arg_x + arg_y + arg_z) * 2;
}

int main(int argc, char* argv[])
{
    int sum = SystemX(2, 2, 2);

    if (sum == 12)
    {
        std::printf("SystemX is working perfectly.");
    }
    else
    {
        std::printf("SystemX isn't working perfectly.");
    }

    return 0;
}
```

Figure 2: System implementation in the form of C++ language functions

It is important to recognize that alterations within a component of a system can significantly impact the system's overall performance. For instance, modifications to the processing pattern or equation, as depicted in Figure 1, can result in a completely altered output. Furthermore, should any part of the system malfunction, it may compromise the functionality of the entire system, ultimately leading to its failure. As illustrated in Figure 2, the performance of a function or system can be assessed by examining its output. This allows for the identification of discrepancies in results, enabling corrective measures to be implemented promptly. Additionally, when developing software that incorporates such functions, it is prudent to employ test cases to evaluate their output. For example, as shown in the subsequent figure, we utilize the boost.test library to scrutinize the performance and output of the function in question. If erroneous results are produced, it suggests a flaw in the logical implementation of the function or system, prompting necessary troubleshooting and resolution efforts.

```
#define BOOST_TEST_MODULE TestAdd

#include "boost/test/included/unit_test.hpp"
#include "../Project/headers/system_x.h"

BOOST_AUTO_TEST_CASE(SystemXTest)
{
    BOOST_TEST(SystemX(2, 2, 2) == 12);
    BOOST_TEST(SystemX(3, 3, 3) == 18);
}
```

```
Microsoft Visual Studio
Running 1 test case...

*** No errors detected
```

Figure 3: System/Function Execution Unit Test

In Figure 3, the function/system is demonstrated to operate effectively for two distinct inputs, subsequently generating our predicted results. In the subsequent section, key characteristics of a system are discussed.

# Types of Systems

---

As previously stated, while any observable entity—such as humans, animals, vehicles, aircraft, computing devices, radar, and sonar systems—can be classified as a system, this delineation is somewhat restrictive. This definition confines our comprehension to tangible and material aspects, neglecting phenomena that, although invisible, are perceivable through their effects or are cognitively recognizable. Nevertheless, all systems within our universe can be systematically categorized as illustrated in Figure 4, where the principal types of systems are depicted.

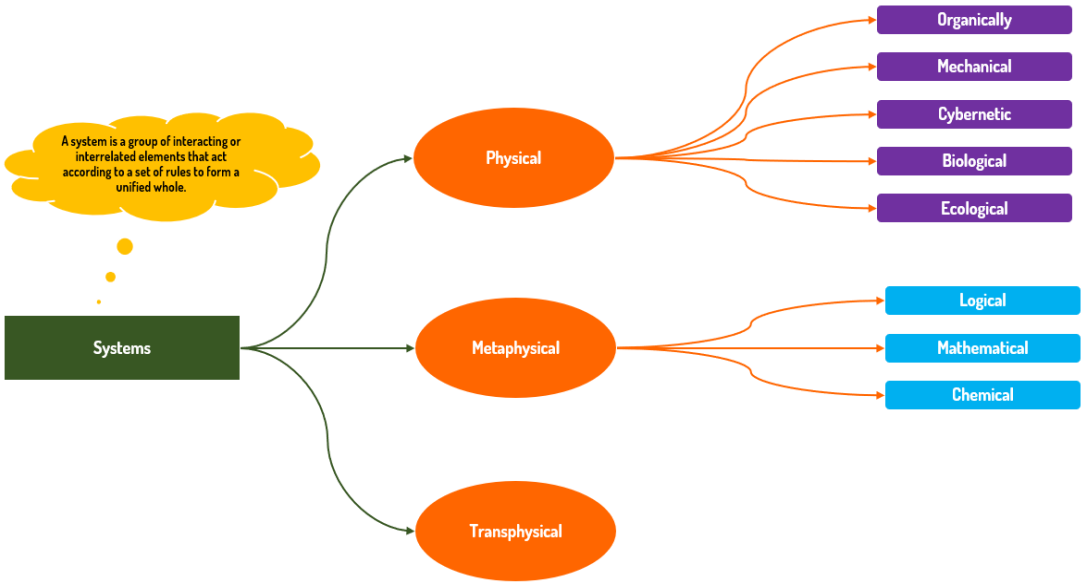


Figure 4: Types of Systems in Existence

In Figure 4, various types of general systems are illustrated. For instance, gravity, though a metaphysical phenomenon, has a profound impact on our lives. Although it cannot be observed directly, its effects can be mathematically described and modeled as a function, which quantifies the force exerted by gravity on Earth based on mass and distance. Similarly, weather, another metaphysical phenomenon, may not be visible, yet its conditions and climate can be accurately predicted by calculating various parameters. Furthermore, software and binary codes represent metaphysical systems; they lack physical form but are instrumental in addressing various computational and practical challenges. An essential aspect of systems and systems thinking is the ability to understand, predict, and infer their performance and future outcomes under different conditions. This involves a comprehensive analysis that extends beyond the immediate attributes of systems, encompassing their interactions and impacts within larger frameworks.

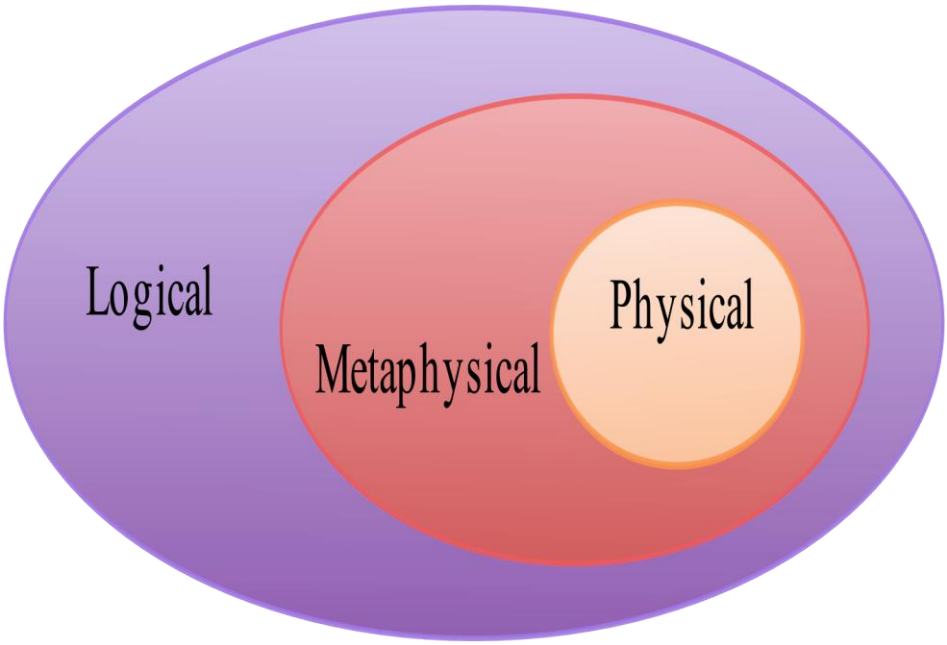


**Figure 5: Nature of Existing Systems**

It is worth mentioning that we can represent any physical or metaphysical phenomenon in the form of a system or function. By doing so, we can examine its past, current, and even future behavior under various conditions or predict its outcomes with a reasonable degree of accuracy. This approach allows us to model complex phenomena in a structured manner, facilitating analysis and understanding. The systems we encounter in existence, however, are typically either purely physical, such as machines, or purely metaphysical, such as gravity, which, despite being an observable force, is more accurately described as a field rather than a tangible entity. Occasionally, we come across phenomena that are Transphysical in nature, such as light, which exhibits dual characteristics, behaving as both photons (particles) and waves. This duality places such phenomena at the intersection of physical and metaphysical domains.

Despite their differences, the behavior of all these systems can be described and explained using the precise language of mathematics. Mathematics serves as a universal tool for modeling, analyzing, and predicting the behavior of systems, whether they belong to the physical,

metaphysical, or transphysical realms. Through mathematical models, we can capture the essential characteristics of these systems and apply logical reasoning to explore their dynamics and interactions. However, for greater precision, it is useful to categorize existing systems into three broad types: logical, metaphysical, and physical systems. This classification allows for a more nuanced understanding of the distinct nature of different phenomena, especially when we set aside Transphysical systems, which are rarely observable and generally confined to the quantum world where classical descriptions break down.

By categorizing phenomena into these three types of systems, we gain a clearer understanding of their distinct characteristics and the methodologies appropriate for studying them. While physical systems can be explored through experimentation and observation, metaphysical systems require philosophical and logical analysis, and logical systems rely on formal reasoning and mathematical rigor. This classification also highlights the importance of interdisciplinary approaches, as the boundaries between these systems are not always rigid. For example, the study of quantum mechanics involves both physical and metaphysical considerations, as well as the application of logical and mathematical frameworks. Similarly, advances in artificial intelligence and cognitive science require the integration of physical, logical, and metaphysical perspectives to fully comprehend the nature of intelligence and consciousness.

In summary, while the behavior of all systems can be described using mathematics, understanding their nature requires recognizing the distinctions between physical, metaphysical, and logical systems. This categorization provides a foundation for exploring the diverse phenomena we encounter in existence, whether they are tangible, abstract, or a complex blend of both.

# Logical systems

Logical systems are abstract and conceptual frameworks designed to formalize the principles of valid reasoning and inference, providing a structured approach to analyzing and understanding the logical relationships between propositions, expressions, or symbols. These systems focus on the rigorous application of logical rules, which allow for the precise evaluation of arguments, the detection of logical fallacies, and the validation of conclusions drawn from premises. By abstracting from specific content, logical systems enable the study of reasoning in a generalized and universally applicable manner.

The purpose of logical systems is multifaceted. They are essential tools for displaying, organizing, and analyzing the complex relationships between different statements, ensuring that conclusions follow logically from given premises. This is achieved through the use of well-defined rules and operations that govern the manipulation of symbols and expressions. Logical systems are foundational in disciplines such as philosophy, computer science, mathematics, and linguistics, where they underpin the development of formal languages, algorithms, proofs, and models of computation.

Examples of logical systems include formal logic, which provides the basis for deductive reasoning and includes systems such as propositional and predicate logic; mathematical logic, which extends these principles to the study of mathematical structures and proofs; and symbolic logic, which emphasizes the use of symbols to represent logical forms and relationships. Each of these systems plays a crucial role in advancing our understanding of reasoning processes, enabling us to construct valid arguments, solve complex problems, and develop reliable computational methods. Through their application, logical systems serve as the foundation for much of modern theoretical and applied science, contributing to advancements in areas ranging from artificial intelligence to theoretical physics.

# Metaphysical systems

Metaphysical systems are abstract frameworks that delve into the fundamental nature of reality, existing at an intermediate level of complexity—simpler than logical systems but more complex than physical systems. These systems focus on answering profound questions about existence, reality, causality, essence, and the nature of being. Metaphysical systems aim to comprehend the ultimate nature of reality and the underlying principles that govern it, providing insights into the fundamental structures of the world that are not directly observable.

The purpose of metaphysical systems is to explore and understand the essential nature of reality, offering explanations for phenomena that lie beyond the reach of physical observation or empirical measurement. For instance, when gravity is considered as a metaphysical system, understanding its principles allows us to address many fundamental questions about the universe. Similarly, other metaphysical systems, such as quantum fields, information and data, electricity, the mind, sound, light, software, and middleware, represent aspects of reality that, while not visible to the eye, are crucial for explaining how the world operates.

These systems play a vital role in philosophy, science, and technology by providing a deeper understanding of the unseen forces and principles that shape our existence. By studying metaphysical systems, we can gain insight into the nature of reality itself, enabling us to address fundamental problems and advance our knowledge in various fields. The exploration of metaphysical concepts is essential for developing a coherent worldview, as it allows us to grasp the intricate and often hidden connections that govern the fabric of reality.

# Physical systems

Physical systems are tangible, real, and observable entities that exist within the natural world. These systems are characterized by their physical presence and are directly accessible to empirical observation and experimentation. The focus of physical systems lies in the study of physical entities, their properties, and the interactions that occur within the observable world. By examining these systems, scientists and researchers aim to uncover the fundamental laws that govern the behavior of matter and energy, as well as the dynamics that shape the physical universe. The purpose of studying physical systems is to gain a comprehensive understanding of

the principles that underlie the physical world. This includes exploring the laws of physics, which dictate how matter and energy interact, and understanding the mechanisms that drive various natural and man-made systems. By analyzing physical systems, we can predict and explain phenomena, develop new technologies, and solve practical problems related to the physical world.

Examples of physical systems include the solar system, which is a complex network of celestial bodies governed by gravitational forces; the human circulatory system, which regulates blood flow and sustains life; and computer systems, which process information through a series of electronic components. Each of these systems operates according to specific physical laws and principles, and their study provides valuable insights into the workings of the universe. By exploring physical systems, we can deepen our understanding of the natural world and harness this knowledge to improve our quality of life and advance scientific and technological progress. In short, logical systems deal with abstract relationships and rules of reasoning. Metaphysical systems ask fundamental questions about the nature of reality. Physical systems include observable organisms and interactions in the natural world.

# Hybrid Systems

It should be noted that the categories outlined in the previous section are not mutually exclusive and may overlap in certain instances. Entities that exhibit such overlaps are referred to as hybrid systems. Hybrid systems are characterized by their ability to span multiple domains—physical, logical, and metaphysical—integrating elements from each to function as a cohesive whole. For instance, a computer system is a prime example of a hybrid system. It consists of a physical component (hardware) that operates according to the laws of physics, and a logical component (software) that adheres to formal principles of computation and logic. However, the functioning of a computer system goes beyond just physical and logical considerations. The exploration of its existence and operations also involves metaphysical considerations, particularly concerning the nature of information, computation, and the abstract algorithms that drive software behavior. In this sense, a computer system embodies both a tangible, physical dimension (hardware) and an abstract, metaphysical dimension (software), making it a classic example of a hybrid system.

Similarly, humans can be considered hybrid entities. They possess a physical body, composed of biological systems that operate within the framework of physical laws. However, humans also have a metaphysical and immaterial component, often referred to as the mind or consciousness, which cannot be fully explained through physical or biological means alone. The mind encompasses thoughts, emotions, beliefs, and self-awareness—attributes that transcend the purely physical and delve into the realms of metaphysics and philosophy. This dual nature of humans, with both a corporeal and an abstract dimension, positions them as quintessential hybrid beings.

Furthermore, hybrid systems are not limited to technology or biology; they can be found in various other domains as well. For example, organizations and societies can be viewed as hybrid systems,

where physical infrastructure and tangible resources are combined with abstract concepts like governance, culture, and values. These systems rely on the interplay between the physical and the metaphysical to function effectively and to adapt to changing circumstances. In essence, hybrid systems challenge the traditional boundaries between categories by demonstrating that complex entities often require an integrated approach that considers multiple dimensions of existence. Understanding these systems necessitates a holistic perspective that recognizes the interconnectedness of the physical, logical, and metaphysical aspects of reality. As we continue to explore and develop new technologies, deepen our understanding of human nature, and navigate the complexities of societal structures, the concept of hybrid systems will play an increasingly important role in our efforts to comprehend and shape the world around us.

# Basic Characteristics of a System

---

These nine characteristics are indeed fundamental to both physical, metaphysical, Transphysical and also hybrid systems, as they define the structure and functionality of any phenomenon considered a system. Below is a more detailed and accurate exploration of these characteristics, emphasizing their significance and application:

1. A system is comprised of multiple components or subsystems:
    - Every system is a composite entity, consisting of various interrelated components or subsystems. These subsystems can range from mechanical components in a machine to abstract concepts in a metaphysical system. The complexity of a system is determined by the number and nature of its subsystems, as well as the relationships between them.
2. These subsystems engage in continuous communication and interaction:
    - The effectiveness of a system depends on the interaction and communication between its subsystems. This interaction ensures the transfer of energy, information, or matter between components, enabling the system to function cohesively. For example, in a biological system, cells communicate through chemical signals, while in a computer system, components exchange data through electrical signals.
3. Every system is situated within an environment:
    - Systems do not exist in isolation; they are embedded within larger environments that can also be considered systems. For instance, an ecosystem contains various subsystems like flora, fauna, and climate, all interacting within the broader environmental system. The environment influences the behavior and functionality of a system by imposing constraints, such as physical laws, resource availability, or social norms. These constraints shape the system's operations and limit its potential actions.
4. Subsystems interact through interfaces:
    - Interfaces define the points of interaction between subsystems, specifying the types of inputs needed for effective communication. For example, in an electrical

system, interfaces determine the voltage and current requirements for components to interact successfully. Just as inputs are defined, so too are the expected outputs. These outputs are the results of interactions between subsystems and are essential for maintaining the overall functionality of the system.

5. A boundary is typically established for the system:
    - Boundaries delineate the limits of a system, separating it from its environment. These boundaries can be physical, such as the outer shell of a machine, or abstract, such as the conceptual limits of a theoretical model. The boundary defines the area within which the system exerts control and influence. It sets the scope for the system's operations and interactions with its environment.
6. Every system operates under a set of constraints:
    - Some constraints are intrinsic to the system, such as the laws of physics governing the movement of mechanical parts or the logical rules governing a computational process. Other constraints are external, imposed by the surrounding environment. For instance, a car's performance is constrained by road conditions and fuel availability, while a software system is constrained by hardware limitations.
7. A system can receive none, one, or more than one input:
    - Systems may operate with varying levels of input. Some systems, like autonomous machines, require minimal or no external input once activated. Others, like interactive software, depend on continuous user input to function effectively. The diversity of input sources and their frequency influences the system's adaptability and responsiveness.
8. A system produces outputs that may not correspond directly in quantity to the inputs received:
    - The relationship between inputs and outputs in a system is not always linear. In some cases, small inputs can generate large outputs (e.g., a small spark igniting a large fire), while in others, significant inputs may result in minimal outputs. This nonlinearity is a critical aspect of system dynamics and can be influenced by factors such as feedback loops and amplification mechanisms.
9. All systems have goals and a system that fails to achieve its objectives is indicative of malfunction:
    - Every system is designed with specific goals or purposes in mind. Whether it's a machine engineered to perform a task or a biological system striving for survival, the achievement of these goals defines the system's success. If a system fails to meet its objectives, it may indicate a malfunction or breakdown, necessitating repairs or adjustments to restore functionality.

# Practical Examples of Systems

Here are some practical examples of systems:

- **Automobile as a System:** An automobile is a system composed of various subsystems, including the engine, fuel, electrical, and suspension systems. These subsystems work

together to enable transportation. If a critical subsystem, such as the engine, fails, the automobile ceases to function as a transportation system, demonstrating the interdependence of its components.

- **Human Body as a System:** Similarly, a human being is a system made up of essential subsystems such as the heart, brain, and sensory organs. These subsystems collectively enable life. The failure of a key component, such as the heart, results in the cessation of life, terminating the system.
- **Computer as a System:** A computer is another example of a system, consisting of multiple subsystems including UEFI/BIOS, RAM, ROM, CPU, GPU, storage (SSD or HDD), networking cards, and the operating system. These components work in harmony to enable the computer's functionality, demonstrating the integration of both hardware and software subsystems.

# The Importance of Synergy in Systems

It is crucial to recognize that a single executive component alone does not constitute a system. A system is defined by the presence and interaction of two or more elements that coexist and work together synergistically to form an identifiable and functional entity. This synergy is the cornerstone of system dynamics, where the collective effect of interacting subsystems is greater than the mere sum of their individual contributions. In other words, the value generated by the system as a whole exceeds what could be achieved by simply adding up the outputs of its separate parts. Synergy occurs when subsystems collaborate and coordinate their activities, leading to emergent properties that are unique to the system as a collective unit. These emergent properties are often unpredictable and cannot be fully understood by examining the individual components in isolation. Instead, they arise from the complex interactions and interdependencies among the subsystems, which amplify the system's overall capabilities and performance.

In the context of complex systems—whether they are biological, technological, or organizational—synergy is fundamental to their successful operation. For example, in a biological system such as the human body, organs and tissues work together in a coordinated manner to maintain homeostasis and support life. The heart, lungs, and circulatory system, while each critical on its own, achieve a level of functionality and efficiency through their interactions that none could achieve independently. Similarly, in technological systems, such as a computer, various hardware and software components interact to perform complex tasks. The CPU, memory, and storage devices each have specific roles, but it is their coordinated interaction—managed by the operating system and other control mechanisms—that enables the computer to function as a cohesive unit capable of executing sophisticated operations.

In organizational systems, synergy is reflected in the way teams and departments collaborate to achieve common goals. When individuals or groups within an organization work together effectively, they create outcomes that are greater than the sum of their parts, such as innovative solutions, enhanced productivity, and improved decision-making. Thus, understanding and fostering synergy is essential for designing and managing systems that are not only functional but also optimized for performance and resilience. The concept of synergy highlights the

importance of collaboration, coordination, and the integration of diverse elements to create a system that is more effective and powerful than the sum of its individual components.

## Practical Example of Synergy in Computer Systems:

For example, consider CPU and GPU collaboration in graphics rendering:

- In a modern computer system, the synergy between the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) is a prime example of how different components work together to achieve superior performance. The CPU is responsible for general-purpose processing tasks, such as running the operating system, handling input/output operations, and managing various applications. The GPU, on the other hand, is specialized for handling complex mathematical calculations required for rendering images, video, and other graphical content.
- When rendering high-definition graphics in video games or during 3D modeling, the CPU and GPU must work in harmony. The CPU handles tasks such as game logic, physics calculations, and managing input from the user. It then delegates the heavy lifting of rendering detailed graphics to the GPU. The GPU processes large amounts of data in parallel, generating the images and visual effects required for the display.
- This collaboration allows the system to render high-quality graphics at a fast frame rate, something neither the CPU nor GPU could achieve alone. The synergy between these components results in a smoother gaming experience, faster rendering times, and overall better performance in graphical applications.

Another example about synergy is Microservices Architecture in the web applications:

- In software systems, a microservices architecture exemplifies synergy. Unlike traditional monolithic architectures, where all functionalities are tightly coupled in a single application, microservices architecture breaks down the application into smaller, independent services that work together to deliver the overall functionality.
- Consider an e-commerce platform that uses microservices for different functionalities such as user authentication, product catalog management, order processing, and payment handling. Each microservice is developed, deployed, and scaled independently, using the best-suited technology for its specific task.
- When a customer places an order, the process involves multiple microservices: the authentication service verifies the user's identity, the product catalog service retrieves product information, the order service processes the order, and the payment service handles the transaction. These microservices communicate with each other through APIs, working together to complete the transaction seamlessly.
- The synergy in this system comes from the fact that each microservice can be optimized for its specific role while still contributing to the overall functionality of the application. This modularity allows for easier maintenance, quicker updates, and better scalability. If the product catalog needs to handle a surge in traffic during a sale, it can be scaled

independently without affecting other services, ensuring the platform remains responsive and reliable. This collaborative operation of microservices results in a robust, flexible, and high-performing software system.

## System Vulnerabilities:

In any scenario, systems are susceptible to threats that can endanger their integrity. These threats may be appeared from two different places which we can categorized them as follows:

1. **Internal threats:** Originating within the system itself, such as component failures, logical errors, or resource depletion.
2. **External threats:** Stemming from outside sources, such as environmental changes, cyber-attacks, or physical damage.

Both types of threats can undermine the identity and functionality of a system, highlighting the need for robust protective measures. For example, in cybersecurity, safeguarding a computer system involves defending against both internal vulnerabilities, such as software bugs, and external threats, such as malware attacks. We will talk about more about these threats in the future. By understanding these characteristics and examples, we gain a comprehensive view of what defines a system, its components, interactions, and the potential risks it faces. This knowledge is crucial for designing, managing, and protecting systems across various domains, from engineering to biology to information technology.

# Computers in the form of a Systemic View

---

The rationale behind classifying computers as systems becomes evident when considering their composition. A computer is assembled from multiple components, each of which must function correctly to produce computational outputs from these electronic and logical machines. Therefore, we refer to them as computing systems or simply computers. For instance, in Figure 6, the array of components that constitute a computer system is illustrated, showcasing how these individual parts integrate to form a cohesive unit capable of performing complex tasks.

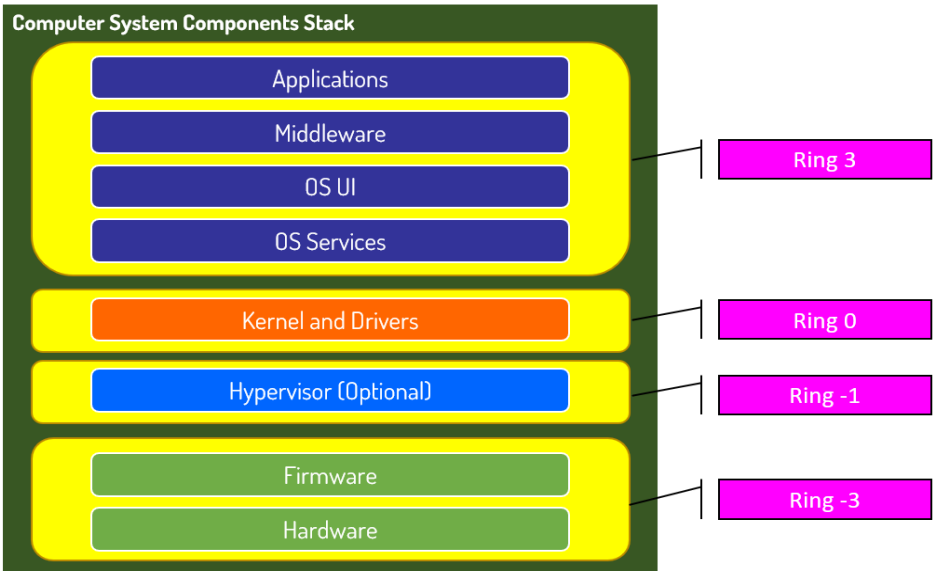


**Figure 6: Computer Subsystems from Organizations View**

According to this definition, a computer qualifies as a system due to its composition of various components or subsystems that collaboratively execute a series of computational, operational, and control tasks. As illustrated in Figure 6, each computer comprises multiple subsystems, with each subsystem specifically engineered to perform distinct functions at various levels. This integration of specialized subsystems enables the computer to efficiently handle diverse and complex processes.

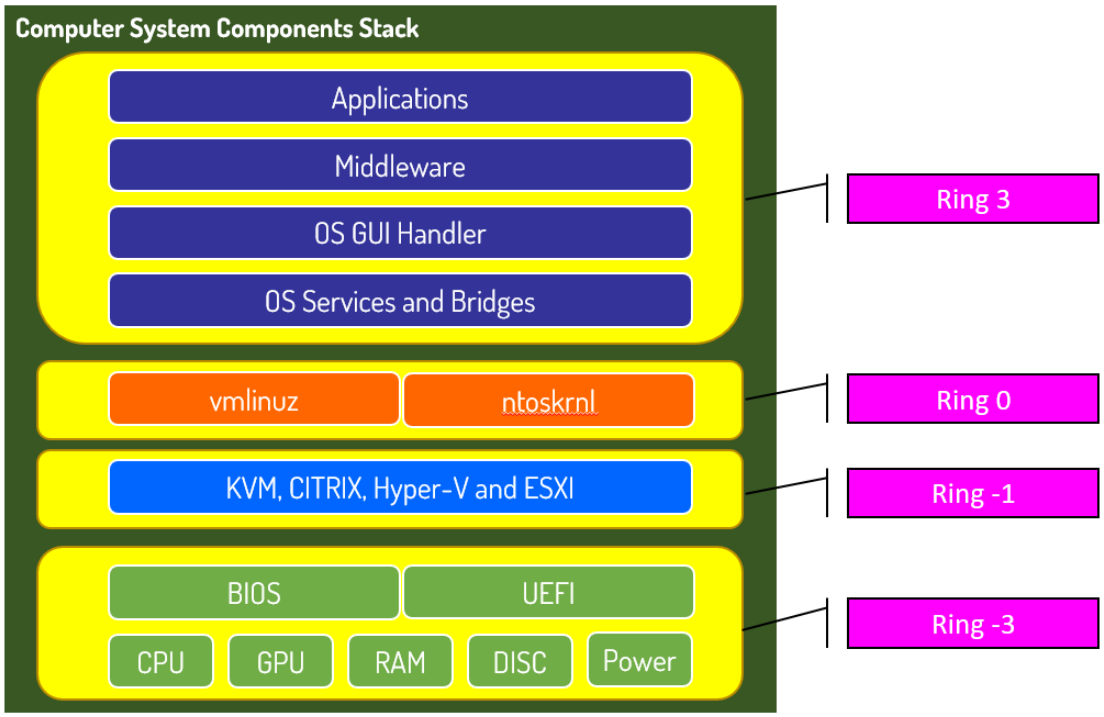


**Figure 7: Operative Components Levels of the Computer Systems**

For instance, if a fault or failure occurs in one of the subsystems at Level -3, as depicted in Figure 7, the operational capability of the entire system is significantly compromised and we can't even power-on the machine at all. However, a malfunctioning component at Level 3 (User-mode) may still allow the system to power on, enabling the operating system's kernel to load and execute. However, the system may not perform optimally, leading to reduced efficiency, slower processing speeds, or even intermittent failures. This degradation occurs because subsystems at Level -3 are typically involved in managing essential hardware components and resources that are critical for the overall system's operation. As a result, any issue within a Level -3 subsystem can propagate through the system, adversely impacting not only its own functionality but also the performance of all subordinate subsystems. This cascading effect is a common pattern observed in both physical and metaphysical systems.

In such systems, the hierarchical structure plays a crucial role in determining the impact of subsystem failures. Subsystems positioned at lower levels in the hierarchy (closer to the core functions of the system) generally have a more significant influence on the system's overall functionality. This is because lower-level subsystems manage fundamental operations, such as hardware control, resource allocation, and basic computational processes, which are vital for the system's stability and performance. In contrast, subsystems positioned at higher levels (closer to the user interface or less critical layers) tend to have a diminished impact when they fail. These higher-level subsystems often handle more abstract functions, such as user interactions, application-specific tasks, or peripheral management, which are less critical to the core operation of the system.

It is also noteworthy that subsystems located lower in the hierarchy typically gain greater access to system resources and possess more operational autonomy. For example, software operating at Level -3, such as bootable device drivers or low-level system utilities, has the ability to influence the entire system's performance by managing core hardware components, memory allocation, and system interrupts. These subsystems operate with a high degree of control over the system's foundational processes, enabling them to significantly affect the overall functionality. In contrast, software operating at a higher level, such as Level 3, generally interacts with more abstracted functions, such as graphical user interfaces or application-level processes, and therefore has limited capacity to directly influence the core system components.

This hierarchical influence is a fundamental characteristic of all systems, whether they are physical systems, like computer architectures, or metaphysical systems, such as conceptual frameworks or organizational structures. Each component within a computer system, for instance, is a system in its own right, often referred to as a subsystem. These subsystems, when functioning cohesively, form a supersystem—what we commonly recognize as a computer. The integrity and performance of the overall system depend on the seamless interaction of these subsystems, each of which plays a distinct role in the system's operation.

For instance, Figure 8 illustrates the various subsystems that comprise the CPU component of the Intel processor's x64 architecture. The CPU, although a single component within the broader computer system, contains multiple intricate subsystems such as arithmetic logic units (ALUs), control units, and various levels of cache memory. Each of these subsystems is responsible for

specific tasks, like executing instructions, managing data flow, or temporarily storing frequently accessed data. The coordination and efficiency of these subsystems are critical for the CPU to perform its functions effectively, which in turn, ensures that the entire computer system operates smoothly.

In summary, the hierarchical structure of subsystems within any system, whether physical or metaphysical, plays a critical role in determining the system's overall functionality and resilience to failures. Understanding the interaction and significance of these subsystems is essential for designing robust systems that can withstand faults and continue to operate efficiently.

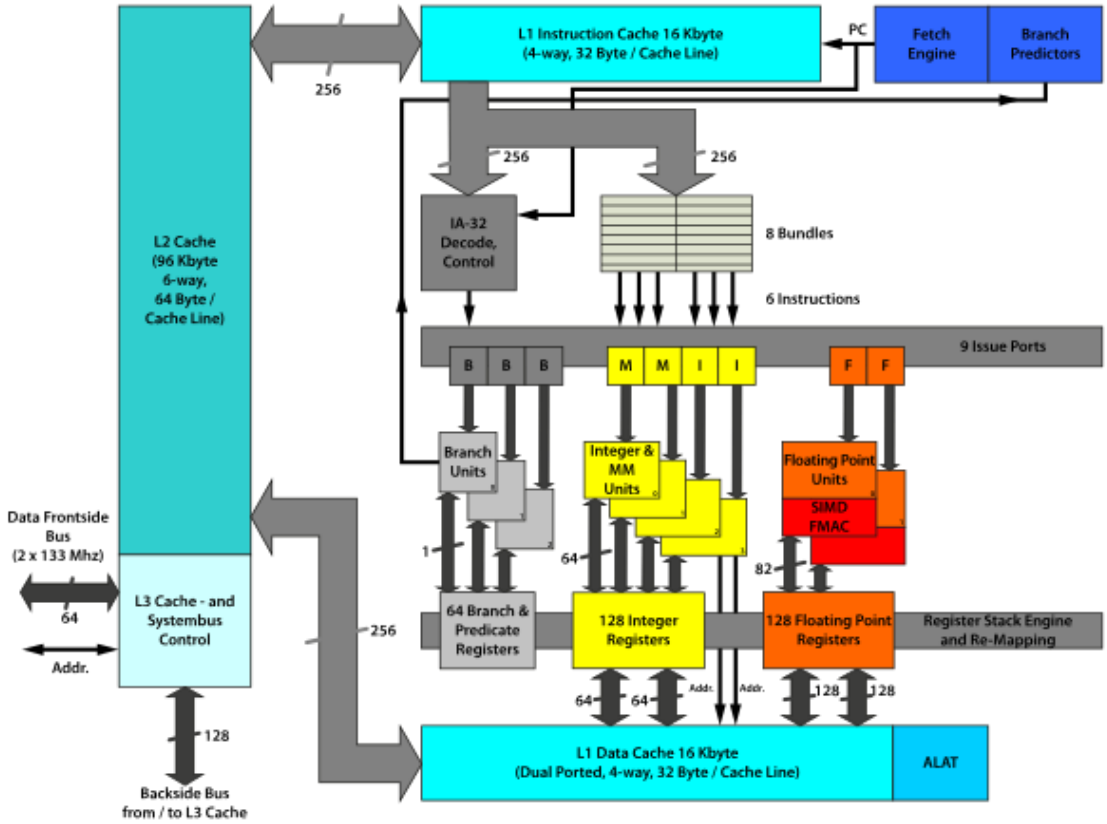


**Figure 8: Subsystems that make up a 64-bit Intel Processor**

A hybrid supersystem, such as a computer, encompasses hardware (physical subsystems), software (abstract/virtual subsystems), and one or more users who oversee and regulate the operations of these subsystems. The hardware component of a computer system typically includes the processor, main memory, hard disk, keyboard, and power source. Conversely, the software component comprises the operating system along with any additional applications installed on the hardware, facilitating user interaction and control over the physical subsystems. As depicted in Figure 6, the fundamental components of a computer system are broadly categorized into hardware, firmware, and software. This tripartite structure ensures a cohesive operational framework, allowing for efficient management and execution of computational tasks.

# Difference Between System and Process

In certain scenarios, individuals may encounter difficulty in discerning between the terms system and process, erroneously assuming them to be synonymous. However, it's crucial to recognize that these are distinct concepts. Broadly speaking, a system entails a collection or amalgamation of abstract or physical components that coalesce to constitute a complex entity or unit.

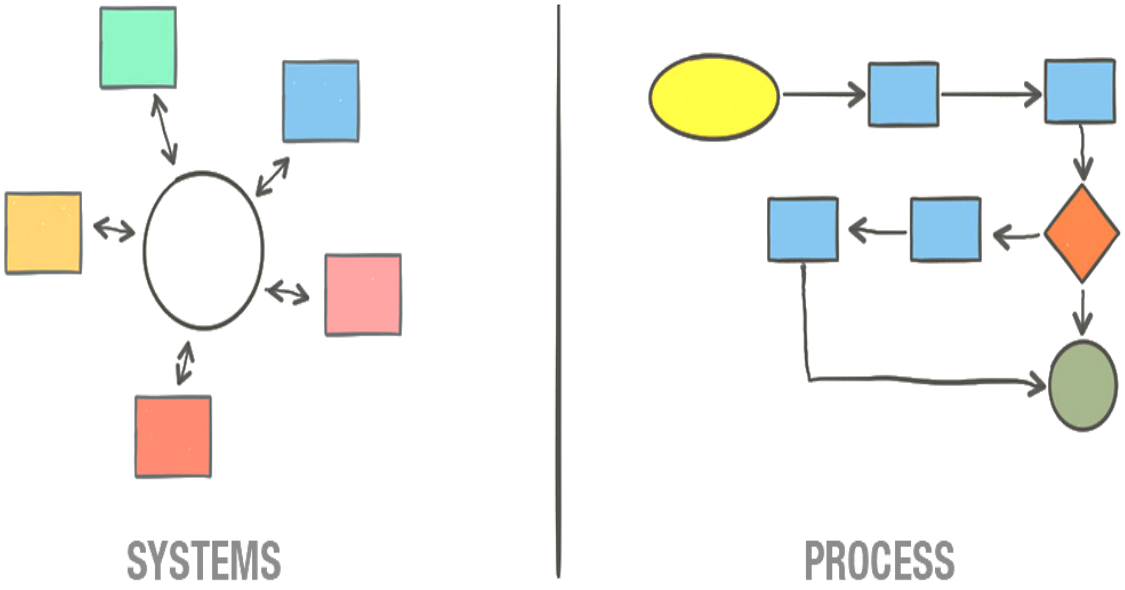


**Figure 9: The difference between a Process and a System**

For example, computers are a good example of a system. A computer has the goal that a user can perform calculation, control, operation, every kind of operations with it. Therefore, a computer system provides many applications for us, including providing a resource for performing heavy calculations automatically, conducting navigation for ships, cars and planes, the possibility of establishing communication between humans, etc. But if we do not have all the subsystems that make up the computer or some of the subsystems do not work properly, the computer system will fail to provide its output. On the other hand, when we talk about processes, we refer to specific flows of activities within a system in order to perform a specific task. Furthermore, we can think of processes as a smaller part of a larger subsystems. It is important that the processes are efficient and transparent in what they do so that the system can effectively succeed in its overall goal. In short, processes are a sequence of activities within a system that are intended to produce a specific result in the form of a system.

**Processes in the form of a computer system link information flows and hardware and software resources together to create and provide output.**

Consider Windows operating system or other operating systems like Unix/Linux or macOS that all of them have different types of process. Each of those process has a well-defined and clear task in the operating system environment. For example, when we are using Windows operating system, the session manager process is the first user-mode process. It's responsible for creating the list of the environment variables, the security descriptors that'll be used by the various system resources, initializing the rest of the registry (HKLM Software and the SAM and security hives), and a lot more. The system process creates the first SMSS instance and it's known as the master SMSS.EXE process. It's the only instance of SMSS.EXE that'll stay after windows have booted up and one of the characteristics of this process is that it doesn't have any command-line arguments.

The master SMSS process will create at least two instances of itself. One in "Session 0" (OS) will be responsible for creating the "wininit.exe" process and another in "Session 1" (User) that'll represent the first logged-on user and will create the "winlogon.exe". Both will spawn a "csrss.exe" process. All the children SMSS.EXE processes created by the master SMSS.EXE process will exit after finishing their JOB.  It is essential to distinguish between systems and processes from a conceptual perspective. When analyzing a system, it is important to first identify its fundamental components and then define the processes within the system, as well as the components involved in carrying out tasks.

---

**For example, when a user attempts to log in to a Windows or Linux operating system, one must consider which processes are involved in handling credentials and conducting authentication and authorization tasks for the user in operating system. After detection of those process, and their relationships we can progress with other analysis and fuzz test approaches.**

---

This concept is crucial for anyone working as a system security researcher or vulnerability analyst. Our primary task involves identifying the key components that constitute a system and mapping out the processes and information flows / information type that enable its functionality. By understanding these elements, we can analyze how information streams can be manipulated to compromise the system's integrity. This approach allows us to explore potential vulnerabilities and develop strategies to exploit them effectively. However, choosing the best fuzzer for testing software at runtime and compile-time after detection the relationship between process and their information flow, depends on various factors including the specific needs of your project, the programming language used, and the type of vulnerabilities you are aiming to discover. Here are some popular fuzzers known for their effectiveness in different scenarios:

1. **AFL (American Fuzzy Lop):** AFL is renowned for its efficiency in runtime fuzzing. It uses genetic algorithms to automatically discover test cases that trigger new internal states in the target binary. This tool is excellent for fuzzing applications written in C and C++ and is widely used due to its ease of use and powerful analysis capabilities.
2. **LibFuzzer:** Part of the LLVM project, LibFuzzer is a library for in-process, coverage-guided fuzz testing of other libraries. It works well with projects that can be linked with LLVM-

based instrumentation. LibFuzzer is particularly effective for compile-time fuzzing, as it requires the code to be instrumented at compile time.

3. **honggfuzz:** A security-oriented, feedback-driven, evolutionary, easy-to-use fuzzer with interesting analysis options. It supports both runtime and persistent mode fuzzing and can be used with binaries written in any programming language. Honggfuzz is capable of using hardware fault detectors such as Intel Processor Trace for coverage feedback on binary targets.
4. **OSS-Fuzz:** Though not a fuzzer itself, OSS-Fuzz is a free service provided by Google that supports continuous fuzzing for open-source software. It integrates well with existing fuzzers like AFL, LibFuzzer, and Honggfuzz, providing a robust infrastructure for comprehensive fuzz testing.
5. **Peach Fuzzer:** This is a framework for generating and performing data fuzzing and is capable of handling both runtime and slightly at compile-time through API hooks. Peach is known for its extensibility and support for creating custom fuzzers for new protocols.
6. **Radamsa:** This is a general-purpose fuzzer that can be used to test any software during runtime. It mutates inputs from given samples and is easy to integrate with other software testing processes.

For the most effective testing, you might consider using a combination of these tools, as each has strengths in different areas of fuzz testing. AFL or LibFuzzer can be good starting points due to their strong community support and effectiveness in uncovering a wide range of bugs.

# Interaction in System Thinking

---

A fundamental characteristic of any system is the interaction and mutual influence among its components. Each component within a system does not operate in isolation but rather engages in dynamic relationships with other components, forming a complex web of interdependencies. By analyzing a single component, one can determine not only how it influences other parts of the system but also how it is, in turn, affected by them. This interconnectedness is what distinguishes a system from a mere aggregation of independent parts. Without these interactions, the collection of components would lack coherence and functionality, existing only as a disconnected assemblage rather than a fully integrated system. This principle is comparable to the dynamics observed in teamwork, where the value of collaboration becomes apparent.

In discussions about teamwork, a common distinction is drawn between a "team" and a "group." Using systemic terminology, a group can be described as a collection of individuals who do not interact or exert mutual influence on one another. In this sense, a group lacks the cohesion and collaborative function that define a true team. A group is essentially a set of people brought together by proximity or purpose but without the relational dynamics that foster collective achievement. Conversely, when the members of a group begin to engage in purposeful interactions—exchanging ideas, sharing responsibilities, and influencing one another's actions—they transition from being a mere group to becoming a team.

The transformation from a group to a team mirrors the formation of a system out of components. This shift occurs when the individuals involved start to actively contribute to the collective effort, creating a network of interactions that enhances the overall performance. These interactions give rise to a phenomenon known as system behavior, where the outcomes are not simply the result of isolated actions of individual components but emerge from the complex and often non-linear interactions among all parts. System behavior, therefore, transcends the sum of the attributes of individual components or the direct consequences of any single modification within the system. Instead, it is a product of the synergistic interplay that occurs when components influence one another in a coordinated manner.

For example, in a functional system—whether mechanical, biological, or organizational—the behavior of the system as a whole is often unpredictable from the properties of its individual components. In an organizational context, this is evident in how a well-functioning team can outperform a collection of highly skilled individuals working independently. The success of the team stems from the collaborative dynamics, such as effective communication, mutual support, and the division of labor, which allow the team to achieve more than the individuals could on their own.

In summary, the distinction between a system and a mere collection of parts lies in the interaction and mutual influence among its components. This interaction is not only the foundation of system behavior but also the key to transforming groups into effective teams. Understanding this principle is essential in fields ranging from engineering and biology to organizational management, where the goal is to create cohesive, functioning systems rather than simple aggregations of parts.

# Borders in System Thinking

---

Almost any system you choose and talk about has a limit; Unless you want to consider the whole universe as a system. The system boundary is not an external reality. Rather, we humans define the boundaries. For example, the system of the Islamic Republic of Iran is a system that has a clear border, and all the issues of this system are defined in the border defined for Iran. Beyond that border, any problem that exists will not be related to the governmental system of the Islamic Republic of Iran. However, many non-systematic solutions arise from the fact that we define the boundaries of our system too narrowly.

In the context of computer software and hardware, the concept of system boundaries is crucial for defining the scope of operations and potential vulnerabilities. A system, whether it pertains to software, hardware, or both, has predefined limits that dictate its functionality, capacity, and interactions with other systems. These boundaries are not inherent properties of the technology itself, but rather constraints imposed by designers and engineers based on specific requirements and objectives. Take, for instance, a computer's operating system (OS). The OS defines a clear boundary for software applications; it manages the hardware resources and provides necessary services for application software. The boundaries of an OS are set to ensure that applications

work within a controlled environment, affecting resource allocation, security measures, and overall system stability.

Similarly, in hardware design, engineers set system boundaries through specifications such as processor speed, memory capacity, and input/output interface limits. These boundaries determine the hardware's capabilities and interact with the software layers to achieve desired functionalities. For example, a processor may have a maximum heat output and clock speed that define its operational limits. Exceeding these limits could lead to system instability or failure, highlighting the importance of respecting and understanding these boundaries.

Moreover, in the realm of networked computing, system boundaries play a pivotal role in defining the scope of network security measures. Firewalls and network protocols establish boundaries that control data flow between systems, thereby safeguarding them against external threats. These boundaries ensure that only authorized users and data packets can enter or leave a network, which is crucial for maintaining the integrity and security of the systems within the network. However, the delineation of these boundaries can sometimes be too restrictive, leading to inefficiencies or missed opportunities for innovation. For instance, a system designed with overly conservative memory management might underutilize available resources, leading to reduced performance. On the other hand, a system whose boundaries are defined too loosely may become prone to security vulnerabilities, as it might allow for unintended interactions between components or expose sensitive information.

In the case of a national system like the Islamic Republic of Iran, the boundaries are geographically and legally defined, influencing everything from governmental policies to technological regulations within the country. These boundaries are clear and encompass all aspects of governance and system operations within the country. Beyond these borders, Iran's domestic policies and system regulations have no jurisdiction, similar to how software designed for one operating system may not operate on another without specific adaptations. Therefore, in both technology and broader systems, the setting of boundaries is a critical task that requires a balance between security, efficiency, and adaptability. It involves understanding the capabilities and limitations of the system and foreseeing how these boundaries impact interactions with external elements. This foresight allows for better preparedness and adaptability in evolving environments, whether dealing with computer systems or national governance.

# The Purpose of the Systems

---

It is also worth noting that a goal is not defined for all systems, and the system may not have a goal. With this argument (and similar arguments) those who read and follow systems theory in general prefer not to raise this case as one of the characteristics of a system. But for those like us who read and learn systems thinking for use in computer security and information security lessons, it is more appropriate to consider the goal and purposefulness as a feature of the system and assume that one or more goals can be defined on many systems.

However, In the realm of computer software and hardware, the discussion of system goals and purposefulness becomes especially pertinent. While general systems theory may sometimes eschew the notion of inherent goals in systems, in domains like computer security and information security, defining and understanding the objectives of a system is crucial. This focus on goals is not arbitrary but stems from the need to assess and enhance the effectiveness and security of technological systems. For instance, when designing a software application, defining clear objectives is fundamental. These objectives might include ensuring data integrity, maintaining user privacy, or providing resilient service under high traffic conditions. Similarly, hardware systems are designed with specific goals, such as achieving a certain processing speed, energy efficiency, or physical durability.

In cybersecurity, the purposefulness of a system is even more pronounced. Security systems are designed with the explicit goal of protecting information and computing resources from unauthorized access and attacks. This involves setting goals such as detecting and mitigating potential threats, maintaining system integrity during an attack, and ensuring quick recovery from breaches. Without clear goals, it would be challenging to design effective security measures or to assess the robustness of the system against threats.

When system analysts approach computer systems, whether it's an individual application, a complete IT infrastructure, or even cybersecurity protocols, they often start by defining what success looks like for that system. This could involve a variety of metrics depending on the system's purpose, such as uptime percentages for network operations, throughput for database systems, or the number of successfully repelled attacks for security infrastructures. For example, consider a database management system (DBMS). Its goals might include handling a high volume of transactions efficiently, ensuring data consistency, and providing robust access controls. System analysts would measure the success of the DBMS by how well it meets these goals under various conditions.

Moreover, the concept of goals in systems extends beyond functionality and performance. In the context of computer hardware, for instance, sustainability and energy consumption have become critical goals due to the increasing environmental impact of technology. Thus, modern hardware is often evaluated not just on its computational capabilities, but also on its energy efficiency and the environmental footprint of its manufacturing and disposal processes. In this way, defining and evaluating goals is not just a theoretical exercise but a practical necessity that influences the entire lifecycle of a system from design to deployment and maintenance.

System analysts, therefore, play a critical role not just in setting these goals based on current capabilities and expectations but also in continuously reassessing and realigning them as technologies and requirements evolve. This dynamic interplay between setting goals and evaluating system performance against these goals is essential for the advancement and security of both software and hardware systems.

# Open and Closed System

Some systems are connected with the world around them and form an open system. While we may see systems that are closed and have no relationship with the world around them. Theoretically, it is the only truly closed system in the entire universe that has nothing outside of it to interact with, and all other systems are considered to be open systems of some sort.

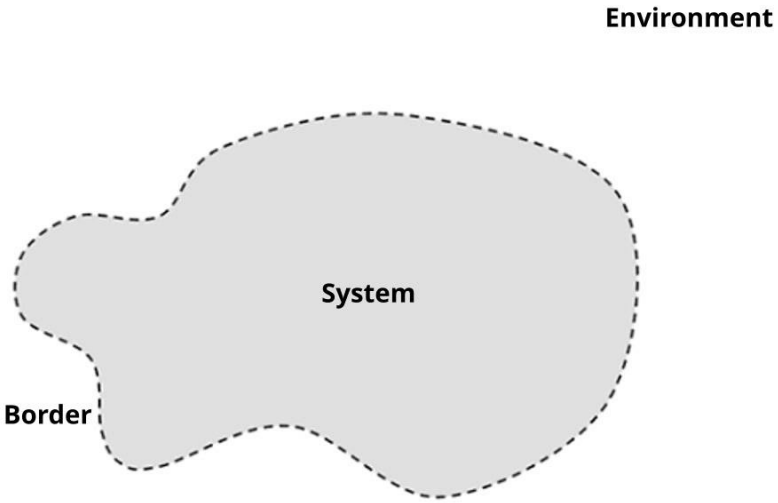


Figure 10: Open and Closed Systems

In the domain of computer software and hardware, the distinction between open and closed systems is foundational and has significant implications for design, functionality, and integration. Understanding these types of systems requires exploring how they interact with their environment, which encompasses everything from user inputs to network connections.

# Open Systems

An open system in computing is one that interacts freely with its environment, capable of receiving and sending information or other resources. This interaction can foster a dynamic exchange that allows the system to adapt, evolve, and integrate with other systems. For instance, a computer connected to the Internet is an open system; it can send and receive data, update its software, access remote resources, and integrate with cloud services. The advantages of open systems include increased flexibility and scalability. They can leverage external resources, such as cloud computing platforms, to enhance performance and capabilities beyond what is possible within their own physical constraints. In software development, open systems are also associated with interoperability and collaboration, as they can work seamlessly with different hardware and software configurations, as well as with other systems designed to be open.

# Closed Systems

Conversely, a closed system in the computing context typically has limited interaction with its environment. This can be by design, to enhance security or stability, or due to operational constraints. An example might be a standalone server not connected to the Internet or a proprietary software system designed to operate independently from other systems. In embedded systems, such as those controlling industrial machinery or medical devices, the systems are often closed to ensure that they operate under tightly controlled variables, minimizing the risk of external disruptions. While closed systems can benefit from increased security and predictability, they often suffer from reduced flexibility and a higher cost of maintenance and integration. For example, updating a closed system can require significant effort and downtime, as it might not be able to automatically receive and install updates or patches.

# Practical Considerations in Systems Analysis

In systems analysis, especially when applied to computing, the terms "open" and "closed" are often used more loosely. A closed system in this context does not mean it is entirely isolated from its environment but rather that its interactions are significantly limited or tightly controlled. Analysts may choose to treat a system as closed if the external interactions are irrelevant to the analysis or if simplifying the system’s model by ignoring external factors makes the analysis more manageable.

For example, when analyzing a network's security architecture, an analyst might treat the internal network as a closed system if the focus is solely on internal threats and vulnerabilities, disregarding potential external attacks. However, such a simplification requires careful consideration, as ignoring external interactions can lead to a failure in anticipating and mitigating real-world threats.

Thus, the decision to view a system as open or closed is not merely a reflection of its actual architectural design but also a strategic choice in analysis. It reflects an understanding of what aspects of the system's environment are most relevant to the goals of the analysis, whether those goals pertain to enhancing performance, ensuring compatibility, or securing the system against threats. Understanding the continuum between open and closed systems helps designers, developers, and analysts make informed decisions that balance the system's functionality, security, and interoperability.

# How does a system work?

Every system may have at least one input, a set of processing-oriented protocols, and also an output or a service. Inputs are items that are used by various processes in the system to achieve the overall goal of the system. In computer systems, data input is raw. These inputs are received from various devices and will be passed to the computer system for processing. In the following, some of the raw data input gates to a computer system are introduced:

- **Keyboard:** The keyboard is included in the category of inputs. The keyboard is used to enter text, numbers and generally a string of letters.
- **Mouse:** After the keyboard, the mouse is known as another way to enter the computer. The mouse is used to move and edit files. Of course, it should be noted that mice can only be used in graphic environments.
- **Scanner:** Scanners are used to convert paper files into digital computer files. For example, you can scan the pages of an article using the page scanner and get a picture of each page of the article on your computer.
- **Modem:** Modems can be placed in both the input category and the output category because it can be used to exchange data and information between computer machines at the level of computer networks.
- **Sensors:** In industrial control systems, PLC equipment receives physical and environmental information from sensors in order to make decisions and control the production process, and will pass it to a PLC for processing.
- **Microphone:** Microphones are used to send audio files to the computer.
- **Webcam:** Webcams, as well as cameras, are another gateway of video inputs to a computer. The visual information of the environment is presented to the computer by the webcam.

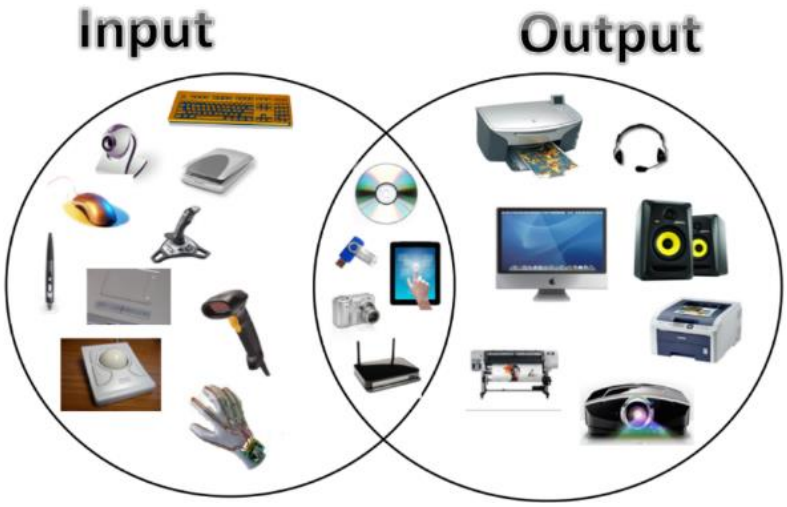


Figure 11: Inputs and Outputs of a Computer System

Also, I should mention here, Processes (at metaphysical layer or software layer) are a set of activities performed by a computer that manipulates and processes various inputs to achieve its defined goal. For example, when the computer receives raw data in the form of sound from the microphone, it can process it and then play it through the speaker, or when it receives raw data in the form of network packets from the network card, it can process that data and send it in Display screen. In picture 10, the input and output equipment of a computer system is shown. Following are the gates that are provided in computer systems for processing or manipulating raw data:

- **Monitor:** The main and most important output of any computer is the monitor. If this part fails, the computer cannot be used anymore. The monitor or the computer screen shows the data processed by the processor graphically to the user.
- **Speaker:** After monitors, speakers are very important when using a computer. Speakers are used to output audio files and various sounds. Since the function of this piece is to play the sound, you can replace it with headphones or earphones.
- **Printer:** Everyone is familiar with printers. These valuable devices print documents and computer image and text files on paper and turn them into paper documents. This device can be considered the opposite of scanning. Printers like scanners are usually used in places that print a large number of files daily.
- **Video card:** The video card is one of the outputs located in the case. The task of the video card is to direct the processed graphics to the monitor. In fact, this device is a complement to the monitor. One of the famous models of video cards are graphics cards, which, in addition to the main computer processor, also have a processor and make changes to the graphics.
- **Actuator:** The function of the actuator or actuator, which is a type of motor in control valves, is to provide the necessary force to open and close the valve and to place it in the desired position according to the signal sent by the controller. Actuators are available in different pneumatic, electric or self-acting types and are selected and used according to the system condition.

Finally, after knowing the input, processing and output rules, we will reach the problem of goals centered on computer systems. Objectives are the final results that the system wants to achieve. In our view, all systems are purposeful and we can evaluate the quality of achieving those goals with the output they produce.

The overall goals of a computing machine are usually described in terms of its mission or purpose. For example, when the goal of a computerized machine is routing, when it receives coordinate inputs from satellites through its antennas, the processing dimension must determine a route on the map. If it doesn't do that, then the computing machine is not working properly compared to its purpose.

# Soft Systems Methodology

Soft Systems Methodology (SSM) was initially proposed by Peter Checkland in 1972 at Lancaster University. The first article outlining SSM, titled "Towards a System-Oriented Methodology for Solving Real-World Problems," was published in the Journal of Systems Engineering in the same year by Peter Checkland. SSM offers a structured and adaptable approach to addressing complex issues. It is noteworthy that the origins of SSM lie in systems engineering, which serves as its foundational discipline. The systems engineering approach involves selecting appropriate means to achieve well-defined goals established from the outset. Peter Checkland categorizes systems into five distinct types: natural, physically designed, abstractly designed, human activity, and transcendental systems. However, it is generally feasible to classify all existing systems into the following categories:

- Physical systems that include the following subsystems:
  - Organic systems such as humans
  - Mechanical systems such as machines
  - Cybernetic systems such as computers
  - Biological systems such as the heart
  - Ecological systems such as hunting
- Metaphysical systems that include the following subsystems:
  - Logical systems
  - Mathematical systems
  - Chemical systems
- Transphysical systems

However, all these systems, no matter what class they are in, will have threats. As mentioned before, everything that is considered as a system is further defined as internal threats and external threats. In the continuation of this article, we will explain some of these threats as well as a standard to measure the security of computer systems.

# What is the threat?

A security threat encompasses any potential risk that may jeopardize the integrity of systems. These threats can manifest in various forms, ranging from physical incidents like the theft of computers containing sensitive data to non-physical intrusions such as malware infiltrations or zero-day based attacks. In the realm of computer system security, a threat signifies a conceivable detrimental action or occurrence, facilitated by a vulnerability, leading to unintended consequences for a computer system or program. Threats can be categorized into targeted attacks, deliberate actions, such as breaching computer systems in a petrochemical plant or

nuclear plants like Duqu or Stuxnet cases, or inadvertent occurrences, like the possibility of a computer malfunction such as a power surge. The realm of cybernetic systems, including computers, is susceptible to a range of threats that pose significant risks to their security. Key among these threats is:

- **Malware:** Malicious software designed to infiltrate, damage, or gain unauthorized access to computer systems. Examples include:
  - **NotPetya:** A destructive ransomware attack that targeted computers primarily in Ukraine but spread globally, causing massive disruptions to businesses.
  - **Stuxnet:** A sophisticated worm believed to be developed by nation-state actors, notably targeting Iran's nuclear facilities by sabotaging industrial control systems.
  - **Duqu:** A sophisticated Trojan used for cyber espionage, believed to be related to Stuxnet and targeting industrial control systems.
- **Side Channel Attacks:** Exploiting unintended channels of communication to extract sensitive information. One example is the use of electromagnetic signals to eavesdrop on cryptographic operations. Side channel attacks exploit indirect information from a system to deduce confidential data or manipulate system behavior. These attacks leverage various aspects of hardware or software operation to gain insights that are not directly accessible. Here are some common types of side channel attacks:
  - **Timing Attack:** A timing attack exploits variations in the time taken to execute certain operations to infer sensitive information. For example, if a cryptographic algorithm takes longer to execute for some inputs compared to others, an attacker can use this timing information to deduce parts of the secret key.
    - **Example:** The time taken to perform a decryption operation might depend on the number of leading zeros in the key. By measuring the time taken for multiple decryption attempts, an attacker can incrementally guess the key.
  - **Power Analysis Attack:** Power analysis attacks monitor the power consumption of a device during computation. These attacks come in two forms:
    - **Simple Power Analysis (SPA):** In SPA, the attacker observes the power consumption directly and looks for patterns that reveal information. For example, certain instructions or operations might have distinct power signatures.
    - **Differential Power Analysis (DPA):** DPA involves statistical analysis of power consumption over many operations to find correlations with secret data.
    - **Example:** In cryptographic devices, different key bits might cause different power consumption patterns during encryption or decryption, allowing an attacker to infer the key.
  - **Electromagnetic (EM) Attack:** Electromagnetic attacks involve capturing the electromagnetic emissions of a device during operation. The emissions can contain patterns that reveal information about the computation being performed.
    - **Example:** An attacker might use an antenna to monitor electromagnetic radiation from a smart card during a cryptographic operation, revealing key bits based on distinct emission patterns.

- **Cache Attack:** Cache attacks exploit the behavior of a system's cache memory. Since cache operations depend on memory access patterns, an attacker can infer information based on which cache lines are loaded or evicted.
  - **Prime+Probe:** The attacker fills the cache (prime), allows the victim to execute, then measures which cache lines have been evicted (probe).
  - **Flush+Reload:** The attacker flushes a shared memory line from the cache, waits for the victim to access it, then reloads it to check if the line was cached again.
  - **Example:** Cache attacks have been used to leak information from shared caches in multi-core processors, particularly impacting shared systems like cloud environments.
- **Acoustic Attack:** Acoustic attacks capture and analyze sound produced by a device during operation. This can reveal sensitive information, as different operations might produce distinct sounds.
  - **Example:** An attacker could record the sounds made by a laptop while it performs cryptographic operations, inferring the key based on variations in fan or processor noises.
- **Thermal Attack:** Thermal attacks exploit the heat patterns generated by a system during operation. Sensitive data can affect heat dissipation, which can be monitored using thermal imaging.
  - **Example:** An attacker might use a thermal camera to monitor the heat profile of a device while it performs cryptographic operations, inferring key bits based on temperature fluctuations.
- **Optical Attack:** Optical attacks use visual signals, such as blinking LEDs or screen brightness, to deduce sensitive information.
  - **Example:** An attacker might observe the blinking of an LED on a router to infer the data being transmitted, as the blinking pattern might correspond to network activity.
- **Fault Attack:** Fault attacks involve intentionally introducing faults or errors into a system to glean information from the resulting abnormal behavior.
  - **Example:** An attacker might expose a smart card to a voltage glitch or laser to disrupt its operation, observing how the card behaves under fault conditions to deduce key information.
- These examples highlight the diverse and often ingenious ways attackers can exploit unintended information leaks or vulnerabilities in hardware and software systems to compromise security.

- **Denial of Service (DoS) Attacks:** Overwhelming a computer system with excessive traffic, rendering it inaccessible to legitimate users. Modern variants include Distributed Denial of Service (DDoS) attacks, which utilize a network of compromised devices. Notable examples include attacks against banking institutions and online gaming services.
- **Man-in-the-Middle (MitM) Attacks:** Intercepting communication between two parties to eavesdrop or alter the messages. This can occur in various contexts, such as Wi-Fi networks or insecure websites.
- **Advanced Persistent Threats (APTs):** Long-term targeted attacks conducted by skilled adversaries, often nation-state actors, with the goal of exfiltrating sensitive data or

disrupting operations. Notable example is Vault 7 which refers to a series of leaked documents published by WikiLeaks in 2017. These documents purportedly originated from the Central Intelligence Agency (CIA) of the United States and provided insights into the agency's cyber capabilities and hacking tools. Here's an overview of some of the key aspects revealed in the Vault 7 documents:

- **Weeping Angel:** This program allegedly targeted Samsung Smart TVs, allowing the CIA to turn them into covert listening devices even when they appeared to be turned off.
- **Marble Framework:** Marble was a framework used to obfuscate the source of CIA malware. It allowed the agency to disguise their cyber-attacks and make them appear as if they originated from another country or entity.
- **Year Zero:** This was the first part of the Vault 7 release and included documentation and details about the CIA's global hacking program. It purportedly contained information about various hacking tools and vulnerabilities in popular software and devices.
- **Dark Matter:** This release focused on hacking tools and malware targeting Apple products, including iPhones and MacBooks. It allegedly included exploits for older versions of iOS and other Apple software.
- **Sonic Screwdriver:** This tool allowed the CIA to bypass the firmware password protection on MacBooks by booting from a peripheral device like a USB drive.
- **HighRise:** HighRise was a tool used to target and exploit vulnerable Windows systems. It allegedly allowed the CIA to intercept and exfiltrate data from compromised computers.
- **UMBRAGE:** UMBRAGE was a repository of hacking techniques and tools collected from other cyber threat actors, including foreign governments and criminal organizations. The CIA purportedly used these techniques to conduct covert cyber operations while attributing the attacks to other actors.

- **Social Engineering Attacks:** Manipulating individuals to divulge confidential information or perform certain actions. Examples include phishing emails, pretexting, and baiting.
- **Brute Force Password Attacks:** Attempting to gain unauthorized access to a system by systematically trying all possible passwords. Attackers use automated tools to execute these attacks efficiently.
- **Supply Chain Contamination Attacks:** Compromising the software or hardware supply chain to introduce malicious components into products. Notably, the SolarWinds supply chain attack compromised numerous organizations by injecting malware into software updates.

It's crucial for organizations to remain vigilant against these evolving threats by implementing robust cybersecurity measures, including regular updates and patches, network segmentation, employee training on security best practices, and the deployment of advanced threat detection and response systems. Each of these threats poses unique challenges to the security of cybernetic systems. For instance, malware encompasses various types, each tailored for specific purposes. Similarly, network attacks, such as eavesdropping, delve into deep intrusions within internal networks, exploiting vulnerabilities at different layers.

# Basic Terms in System Security

In the continuation of this article, in order to understand the proposed standards for network and computer system penetration testing from the perspective of a system such as the PTES standard, we will investigate and study specialized terms and vocabulary in order to familiarize ourselves with the literature of this field.

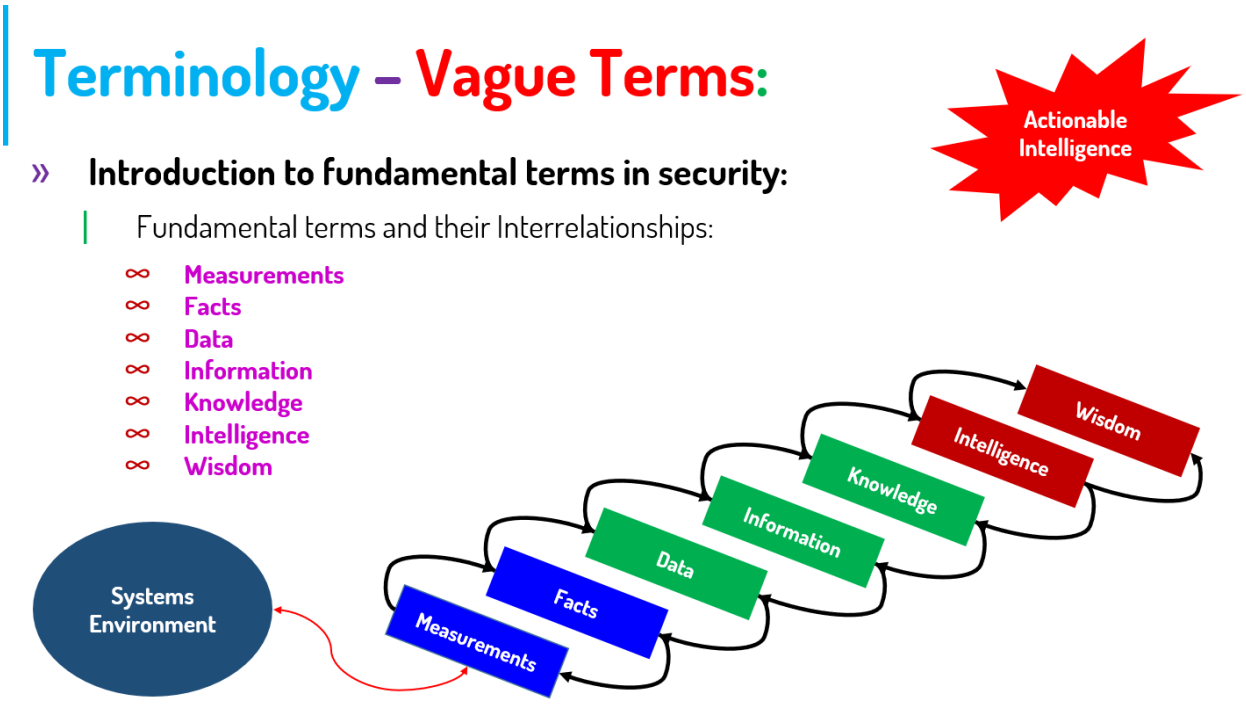


Figure 12: Pyramid of Specialized Vocabulary in Security and Information

In figure 12, which is a part of the slides of Ai000 Cybernetics QLab Practical Malware Analysis course, the specialized vocabulary pyramid in the field of system security is displayed. These terms (specialized vocabulary) are described below:

- **Measurement:** Measurement refers to the quantification of attributes of an object or event within a specific environment, facilitating comparisons with other objects or events. It's important to note that measurements vary significantly across different environments. For instance, the unit of measurement for gravity differs from that of the destructive power of a hydrogen bomb due to the distinct environments and contexts.
- **Fact:** Facts are pieces of information or perceptions that have been verified. For example, gravity is a fact because its presence is universal and undeniable. Similarly, the equation E=mc2 is a fact as it establishes the equivalence of mass and energy.

- **Raw Data:** Raw data, devoid of any processing or classification, exists in all forms, usable or non-usable, and holds no inherent significance. For instance, the sequence 14010630 is a numerical raw data that lacks inherent meaning.
- **Information:** Information is derived from raw data through processing and contextualization. For example, the raw data 14010630, when interpreted as a date, gains meaning and relevance. Information, essentially processed raw data, answers questions like "who," "what," "where," and "when" within a given context. It serves to facilitate decision-making, problem-solving, or opportunity realization. For instance, the number 14010630 could refer to the date of Milad Kahsari Alhadi first session malware analysis course.
- **Knowledge:** Knowledge is a well-organized collection of information intended to be useful. Information transforms into knowledge when it enhances our comprehension (knowing what), capability (knowing how), and understanding (knowing why) of a phenomenon.
- **Intelligence:** Intelligence necessitates the ability to comprehend the environment, make decisions, and control actions. Advanced levels of intelligence encompass the ability to recognize objects and events, represent knowledge in a comprehensive model, and reason about future plans. Intelligence enables successful functioning under varying conditions to survive, thrive, and reproduce in complex and often hostile environments.
- **Wisdom:** Wisdom implies the ability to predict, such as forecasting a person's reaction in a specific situation. When intelligence about a phenomenon is validated across multiple processes, wisdom about that phenomenon or subject is attained.

# Final Words

In this research paper, our team at ACQL has embarked on an exploratory journey to thoroughly understand general systems concepts, aiming to extrapolate these concepts to broader system typologies. Our initial findings suggest that systems, universally, exhibit 9 fundamental characteristics. Furthermore, we identify that any entity defined as a system is susceptible to both internal and external threats, necessitating robust protective measures. Our analysis progresses into a detailed examination of various data terminologies, along with their types and states, focusing primarily on concepts such as information and intelligence. This foundational understanding of data dynamics has enabled us to advance our study into high-level analyses of software-based systems and binaries. By doing so, we have been able to identify potential vulnerabilities within these systems. The culmination of our research involves developing strategies to exploit these identified weaknesses effectively, thereby gaining control over the systems in question. This comprehensive approach not only enhances our understanding of system security but also contributes to the development of more secure computing environments.