MATTER Journal of New Materialist Research

Universitat de Barcelona
http://revistes.ub.edu/index.php/matter/about

Cybersecurity and Simondon's Concretization Theory: Making Software More Like a Living Organism

Ziyuan Meng
Drew University

Jon K. Burmeister
College of Mount Saint Vincent

DOI: https://doi.org/10.1344/jnmr.v3i1.38956

Abstract

The cybersecurity crisis has destabilized the field of informatics and called many of its foundational beliefs into question. This paper argues that Gilbert Simondon's theory of the origin and development of technical objects helps us identify faulty theoretical assumptions within computer science and cybersecurity. In particular, Simondon's view is that the process of the 'individuation' of technical objects can have similarities with the development of living beings – a view that stands in stark contrast with hylomorphic and reductionist views of technical objects currently common in computer science. We argue that those common hylomorphic approaches to software development lead to excessive modularity in software applications, which in turn results in less secure systems. To investigate a new ontological basis of software security, we look to Simondon's ontology to reconsider what makes a piece of software vulnerable in the first place, and we focus on two concepts in his general theory of ontogenesis – 'individuation' and 'associated milieu'. By examining a case study of a malware infection attack, we show that the event of a cyberattack unleashes a 'co-concretization' process of software applications and their associated milieu, namely, their operating system. Both the application and the operating system evolve from an abstract form to a more concrete form by re-inventing their own interiors and re-orienting their relationship to each other. We argue that software development will be more secure if it takes inspiration from the development of living

beings and refocuses on the dynamic reciprocal relationship between software applications and their technical and social environment.

Keywords

Individuation; Information; Cybersecurity; Simondon; Associated Milieu; Software.

Introduction

What does cybersecurity have to do with the highly abstract ontological categories of form and matter, universal and individual? The eminently practical domain of computing and cyber defense might appear to be miles away from the heady heights of metaphysics and ontology. Yet any attempt to understand technical beings and technical processes in a foundational way will go astray if it is founded upon a faulty theory of being. We wish to argue that the current dominant mode of thinking about computers and cybersecurity is in fact based on such a faulty theory of being, and we look to Gilbert Simondon's ontological categories for a stronger account. Simondon's critique of hylomorphism (viewing objects as a combination of form and matter; Simondon, 2009a, p. 4), along with his quasi-biological conceptualization of technical evolution, shed light on the mode of existence of technical beings not as stable individuals with inherent properties, but as a dynamism of restructuring operations which are entangled with their environment. This new temporal, relational ontology of technical beings shares an anti-substantialist theme with new materialism's concept of intra-action (Barad, 2003).

Simondon's unique approach to technical beings is rooted in his deep critique of the alienation between culture and technicity (Simondon, Malaspina, & Rogove, 2017, p. 15). His exhortation to re-integrate technical beings into the web of meaning in the cultural sphere resonates with new materialism's post-anthropocentric stand on harmonious modes of human and nonhuman relationality. The cybersecurity domain can borrow these concepts to critically examine the hylomorphic, reductionist paradigms in computer science and software engineering practice with respect to their effects on software security. The ultimate aim of this paper is to explore a non-instrumentalist, techno-social normativity for cybersecurity research. Software

developers and security engineers can view the maturation of software applications as a process of orienting toward a greater degree of integration, one which resembles the development of living beings.

To develop this argument, in the second section, we will provide an overview of Simondon's theory of 'individuation' in its most general sense – that is, his theory of the process through which an individual develops itself over time (Simondon and Adkins, 2020). We focus on his critique of what we call 'naive hylomorphism,' and examine his view that the traditional concepts of form (*hyle*) and matter (*morphe*) are too static to explain the dynamic phenomena of change and development seen in the world. More specifically, we show how Simondon – in a foreshadowing of the new materialism movement – critiques the common idea that form is the sole active and shaping force in a thing's development while matter is merely passive, inert, and what is shaped. Building on this, we explain the emergence first of vital (biological) individuation, and then of psycho-social individuation (third section). We then take up Simondon's theory of the individuation of *technical* objects (fourth section), focusing on his concept of concretization, i.e., the development of an object from a more abstract stage to a more concrete stage, using an air-cooled engine as the central example. This leads into a discussion of his concept of 'associated milieu,' as exemplified in the traction motor in a train.

In fifth section, we move to the realm of computing and explain how a Simondonian ontology illuminates the individuation of operating systems and software applications as potentially mimicking some elements of the individuation of living beings. We first provide a brief overview of the history of operating systems to reveal the hylomorphic ideology built into the design of modern digital systems. We then use a case study of a malware attack to illustrate that a software application can – like a biological being – become more individuated, more concrete, and more coupled with its technical running environment precisely through being attacked and then responding to that attack.

Simondon's Theory of Individuation

French philosopher Gilbert Simondon (1924-1989) is a unique figure in intellectual history for his lifelong dedication to both philosophy and a detailed, hands-on study of technology. His *Individuation in Light of Notions of Form and Information* (Simondon & Adkins, 2020) presents his general ontological theory of individuation -- how individuals come into being. In this section, we provide a brief introduction to his theory of individuation, explaining four key concepts to prepare for a later discussion of his theory on technical individuation: pre-individual, transduction, information, and associated milieu.

Simondon's main target of critique in *Individuation in Light of Notions of Form and Information* is hylomorphism, a doctrine of being originating in Aristotelian metaphysics, which explains the constitution of an individual being as the union of its 'matter' and its 'form.' In our view, Simondon's critiques apply less to Aristotle's own highly sophisticated hylomorphic theory and more to a simplistic form of hylomorphism which developed later in western philosophy, which we will call 'naive hylomorphism.' This naive hylomorphism understands matter to be a completely passive element of a thing, a pure potentiality to become something different. Under this view, form is the organizing principle which actualizes the potentialities of the matter upon which it is acting. A simple example is a wooden mold (form) used by a brick-maker to shape clay (matter).

For Simondon, however, an individual can rarely exist in a finished form that completely exhausts the potential of its materiality, in part because this materiality is not in fact something purely passive. Rather, an individual is always in the process of inventively developing itself through both its form and its matter, a process which he called *individuation*. If we closely follow the technical operations involved in brickmaking, we see that a sharp distinction between form and matter cannot be maintained. The mold is not a pure form, nor is the clay formless matter. Rather, the mold must be prepared as a form that has its *own* materiality. "In order to produce a form, one must construct a certain defined mold, prepared in a certain fashion with a certain type of matter" (Simondon & Adkins, 2020, p. 23). The clay is matter that *already* possesses a certain form, which is then given a new and different form. Then in a heating process, there is an energy exchange between clay *and* mold. Contrary to

naive hylomorphism, the material of the clay is not passive in this process, but rather expands toward the wall of the mold. The wooden wall of the mold reacts to oppose the pressure from the clay. This exchange of force and energy makes clay into a hardened shape of brick. In other words, what gives rise to the individuation of a brick is the material activity of both the mold and the clay, and the *resolution* of the tensions between those two initially disparate domains of potentialities.

Ingenuity of human labor does play its own part in the emergence of brick, but that labor must work in cooperation with the inventive operations already immanent in both the clay and the mold to bring about a real change. The language of naive hylomorphism cannot grasp this active character of matter; it can only think of an abstract matter and abstract form which are already individuated. To provide a more adequate account, Simondon's theory of individuation emphasizes the dynamic, continuous process in which various forms of individuals emerge from relations between multiple fields of potential. In this process, matter is no longer 'a pile of dead stuff.' Rather, it provides a profound creativity to processes of change, along with the capability of problem solving.

Broadly speaking, Simondon's individuation theory is intended to give ontogenetic accounts of beings at different levels. These include physical, biological, psychic, and social levels of individuation. In all forms of individuation, the same material, operational "formula" is at work. Individuation begins with a system in a primitive state of being called 'pre-individual.' A system in its pre-individual state is abundant with potential and yet does not have a distinct identity. It is "more than unity and more than identity" (Simondon, 2009a, p. 6). That is to say, it contains the potential to begin transformation in multiple directions. Here, Simondon's primary source of inspiration for the concept of pre-individual state of being came from the notion of metastability in thermodynamics. A metastable system is the one that is in an equilibrium which is neither completely stable nor completely unstable. As Muriel Combes summarizes, "a physical system is said to be in metastable equilibrium (or false equilibrium) when the least modification of system parameters (pressure, temperature, etc.) suffices to break its equilibrium" (Combes & LaMarre, 2013, p. 3).

The ontogenetic process begins after an event introduces disparity into the metastable pre-individual system. Then the system begins a series of phase

transitions to resolve the disparity. In each phase of this process, a distinctive kind of individual emerges in the system. This individual, in its structure, preserves some of the potential from the initial pre-individual state, and thus is capable of further individuation.

The growth of crystals is Simondon's paradigmatic example of an individuation process springing out of a metastable pre-individual system. The genesis of a crystal begins in a supersaturated liquid known as a 'mother liquor.' The introduction of a 'germ' – such as a particle – disrupts the equilibrium and polarizes its surrounding mother-liquor. This singular event triggers the process of crystallization, "starting from a tiny germ, increases and extends following all the directions in its supersaturated mother liquor: each previously constituted molecular layer serves as the structuring basis for the layer in the process of forming; the result is an amplifying reticular structure." (Simondon and Adkins, 2020, p. 13)

Naive hylomorphism cannot explain the emergence of such a crystalline structure. The material operation involved in crystallization can hardly be accounted for in terms of a force imprinting a form onto inert, passive matter. In contrast, Simondon's richer conception of matter can make sense of this process, by noting that the operation propagates and amplifies its activity through the very structure it is creating. Simondon replaces the overly simplistic duality of active form and passive matter with the more nuanced duality of 'operation' and 'structure,' both of which actively contribute to the development of the individual.

Borrowing a term from physics, biology, and electrical engineering, Simondon refers to the interaction between structure and operation as *transduction*. In electrical engineering, an example of a transducer is a device which translates energy from one form to another, such as an antenna, which translates radio waves into electrical signals. Simondon ontologizes the general concept of transduction to describe the duality and interplay between operation and structure that is present in any individuation process:

> By transduction we mean a physical, biological, mental, or social operation through which an activity propagates incrementally within a domain by basing this propagation on a structuration of the domain operated from one region to another (Simondon and Adkins, 2020, p. 13).

For Simondon, the concept of form in hylomorphism cannot adequately account for the transductive process, so he proposes that "the notion of form must be replaced with that of information" (ibid., p. 16). By information, he does not mean encoded messages passing from a sender to a receiver through an established communication channel, a narrowly defined concept which came out of the early cybernetics movement. Rather, his notion of information has an operative sense of *in-formation*, namely, the operation of form-taking! In the example of the brick-making process, mold and clay initially exist as two disparate systems. Each system is metastable with the potential to be *deformed*. When the two join together in the heating process, they begin to act upon, or *in-form*, each other. In their transductive exchange, the individuality of the brick emerges when the two systems are eventually stabilized in a new equilibrium. As Simondon puts it, "Information is therefore a primer for individuation" (Simondon, 2009a, p. 10). This reconceptualized notion of information is clearly different from the early cybernetic model of information, which is inherently substantialist in assuming the unchanging individuality of sender and receiver. This model of communication claims that there is a one-to-one correspondence between two individuals that does not structurally change either of them; yet, this is a mathematical myth. The operation of information only emerges in the exchange between "two different orders that are in a state of disparation" (ibid., p. 9) – that is, a state of disparity. Metastability and disparity are the conditions of information. In a real information exchange, as Andrea Bardin summarizes, "there is no univocal transmission, nor a one-to-one correspondence between the systems, but rather we have a concurrent reciprocal influence, and therefore a macro-system composed by A, B and their interaction" (Bardin 2015).

The Theory of Vital and Psycho-Social Individuation

The individuation of *living* beings is based on the same 'interplay of operation and structure' as seen in the individuation of merely physical beings (such as crystals). What is different about the individuation of living beings is that they can do more than merely adapt to their milieu in an external way. A living being actively invents its own exteriority and *interiority*: "the living being solves problems not only by adaptation, that is, by modifying its relation to the milieu, but by modifying itself, by inventing new

internal structures, by inserting itself completely in the axiomatic of vital problems" (Simondon and Adkins, 2020, p. 28). In contrast, during growth of a crystal, the individuating activity only occurs at the ever-expanding surface of the crystal. Its interior does not participate in further individuation. That is, within the crystal there is no true interiority:

> ...the physical individual has no veritable interiority; on the contrary, the living individual has a veritable interiority because individuation takes place from within; inside the living individual, the interior is also constitutive, whereas in the physical individual only the limit is constitutive (ibid., p. 8).

With the invention of interiority, living beings begin to possess the autopoietic character, a power to differentiate the internal and the external. This can be observed in even the most basic form of living being such as a unicellular organism. The membrane selects which elements can be integrated into the interior, and which cannot (ibid., p. 250).

In addition to facilitating the emergence of individuals, the initial pre-individual milieu itself also goes through substantial change. Individuation "does not break the system" (ibid., p. 53) into an individual and a leftover milieu exhausted of its potential. Rather, it introduces a new individual-milieu relation. In the individuation of living beings, there is the emergence of an *associated milieu* as the complement of the living individual. For a living being, its associated milieu is a pathway connecting its interior to a greater domain of being. Through its associated milieu, a living being is able to conserve and renew some remaining potential from the initial pre-individual milieu and carry on the individuation at its own pace:

> ...the principle of individuation…is the complete system in which the genesis of the individual takes place; that, moreover, this system outlasts itself within the living individual as a milieu associated with the individual in which individuation continues to take place (ibid., p. 51).

Moreover, an associated milieu is not something pre-given. Rather, it is invented by a living individual as "a synthetic grouping of two or several levels of reality without intercommunication before individuation" (ibid., p. 383). Simondon often describes the relationship between a living being and its associated milieu as possessing

recurrent causality. The living being creates its associated milieu which, in turn, conditions its existence.

With this view of vital individuation in mind, we can see how *psychic* (psychological) individuation is based upon it. Here, it is important to know that Simondon rejects any substantial separation between the psychic and the vital, emphasizing the continuity between two regimes of individuation. To continue its individuation, a living being engages in vital activities to regulate the relations between its interior and exterior milieu and between what is already individuated and the pre-individual potential. Simondon believes that psychic reality arises from these activities to perpetuate the vital individuation. For example, perception emerges from the vital activities to resolve the conflicts which it encounters with the surrounding milieu. 'Affection' emerges from the vital being's effort to coordinate sense perceptions and actions, and from its effort to regulate the individual-milieu relationship.

The introduction of the psychic domain helps the vital being to maintain its resonance with its milieu, therefore prolonging the vital individuation. But it also poses new problems which a psychic individual cannot solve within itself. When perceptivity and affectivity become incompatible, a psychic individual cannot resolve the problem within itself. It must participate in and be integrated with the individuation of the broader milieu of the collective (i.e., the social) to resolve the tensions and to continue its own individuation. For Simondon, the individuation of psychic beings and individuation of their collective are two poles of *one* process. Between the individuation occurring in the interior of psychic beings and the individuation of their collective milieu which exceeds the individuals, there is a fundamental unity which he calls the *transindividual* relation.

Above we have examined Simondon's theory of individuation in three levels: the physical, the biological, and the psycho-social. He believes that these three levels, in combination, constitute 'nature' as a whole. Now we are prepared to consider how his theory of *technical* individuation relates to and builds off of these concepts.

The Theory of Technical Individuation

Naive hylomorphism is inadequate for understanding the ontogenesis not only of physical, vital and psycho-social beings but also of technical beings. The process of engineering a technical being is often viewed as a direct imposition of a model in the mind of the designer onto inert material elements. Simondon rejects anthropocentric views of technology which tend to reduce technical beings to instruments. For him, an ontological theory of technical objects must trace the dynamical evolutions that take place in their technical lineages through their own necessity and normativity. As Jean-Hugues Barthélémy (2015, p. 20) points out, the individuation of living beings provides the model to reason about the process of technical evolutions.

Levels of Technical Reality

Simondon categorizes technical objects into three levels of existence: the *element*, the *individual* and the *ensemble*. Springs, screws, and transistors are examples of technical elements – simple tools. Technical elements are the carriers of immediate technical operations. A technical element is "free" and "universal" in the sense that it can be integrated into any technical system. When technical elements are organized into a system, the result is a technical *individual*. For instance, an engine is made up of multiple elements or tools. A technical *ensemble* comes into being when multiple technical individuals coordinate through a communications network. An example of a technical ensemble is a factory made up of multiple machines connected via a communications network. But in Simondon's theory of technical evolution, technical individuals are the central focus. At this level of technical reality, the evolution of machines demonstrates an orientation toward a structure which resembles organic beings (Simondon, Malaspina, & Rogove, 2017, p. 60).

Concretization

Concretization is the most important concept in Simondon's theory of technical individuation. This concept describes how a technical object evolves from a more abstract stage to a more concrete stage, i.e., toward coherent technical individuality.

In the abstract stage, different elements of the technical object are linked together but are not fully integrated. The object becomes more concrete when its elements take on a higher degree of structural and functional convergence. Simondon illustrates these concepts by tracing the genealogy of the combustion engine. He observes that in a more primitive engine, "each element intervenes at a certain moment in the cycle, and then is expected no longer to act upon the other elements" (ibid., p. 27). By contrast, in a more advanced engine, "each important item is so well connected to the others via reciprocal exchanges of energy that it cannot be anything other than what it is" (ibid., p. 26).

For Simondon, a higher degree of integration within a technical object has genuinely practical consequences. This greater integration perfects that object by making it more autonomous and more secure. Simondon's language on this point is noteworthy: he speaks of an object's greater integration leading to the emergence of "defense structures" within that object. For example, in the early air-cooled engines, the cooling fins were attached to the engine's cylinder from the outside and only served one function: cooling. The two systems, the cylinder and the cooling system, functioned independently and this meant that the engine lacked integration. But in a more advanced engine, the cooling fins are more integrated with the overall structure of the cylinder because the fins play more than one role. They act as fins to lessen the heat generated by the cylinder but they also act as ribs that "resist the deformation of the cylinder head under the pressure of the gasses" (ibid., p. 27). Concretization is thus the process in which technical beings evolve "into a system that is entirely coherent within itself and entirely unified." (ibid., 2017, p. 29) In this process of evolution, concretized technical objects come to partially resemble natural living beings, even if they can never be identical to the living beings (which are the original concreteness).

Associated Milieu

Concretization increases the internal coherence of the technical object and creates a surrounding milieu for it. This milieu enables a technical object to simultaneously adapt to both the technical and the natural environments in which it is operating. It consists of elements from the natural milieu grouped and synergized with the technical components to support its function. A river dam needs an artificial lake or

reservoir to store water for electricity generation. Simondon calls the kind of milieu that is integrated with a technical object an 'associated milieu.' The associated milieu of a technical object is the necessary condition of the very possibility of its existence. However, Simondon is clear that an associated milieu is not merely a means of adaptation for a technical object. An associated milieu must be invented for a technical object and is also conditioned by it. Simondon describes this co-conditioning relationship between technical objects and their associated milieu as *recurrent causality* (ibid., 2017, p. 60). He uses the example of a traction motor to illustrate the concept of recurrent causality between a technical object and its environment. A traction motor is an electric engine used to propel a train along the track, and it must maintain the speed of a train in as constant a manner as possible as the train travels over a variety of geographical terrains: contours, elevations, sharp turns, etc. A traction motor not only converts electrical energy into the mechanical forces that pull a train on the track, but also adjusts the supply voltage depending on the resistance it receives from the natural environment. In a traction motor, a part of its technical interior must be synergized with the external geographical environment to enable the reciprocal relationship. This relation between the motor and the environment is the associated milieu of a traction motor, the condition of its function (ibid., pp. 55-56).

Although Simondon certainly does not equate concretized technical beings with natural living beings, he sees important similarities between the evolution of technical individuals and the evolution of living beings. Like biological evolution, technical evolution tends toward greater integration by resolving the incompatibility among disparate elements. Like living beings, technical beings stand in relation to an associated milieu. And like living beings, technical beings develop into more mature forms out of an internal necessity contained with them. In other words, for Simondon concretization is not ultimately due to social and economic factors. The functional convergence during the concretization process takes place, in part, due to the non-human factor of an *internal necessity* of the technical being itself (ibid., p. 29). In fact, some human interventions are actually detrimental to a technical object's concreteness. Simondon observes that when many customer requests are imposed on the design of a car, "its essential characteristics are encumbered with external servitude" (ibid., p. 30). In other words, the technical object in such a case has become

less integrated, due to humans failing to bring forth the potential unities that lie within the materials with which they are working.

The reason why Simondon insists upon a non-anthropogenic account of technical evolution can be found in his overall view of alienation in modern age. For him, culture has alienated technicity by reducing it to the domain of mere *usage*, or instruments. Built into culture is a systematic defensive attitude which rejects the possibility of technical beings providing any cultural significance. Simondon found this wall between *kingdom of ends* and *kingdom of means* to be a fundamental blockage in resolving the alienations between humans and alienation between human and nature. He calls for a new technical culture that can "incorporate technical beings in the form of knowledge and in the form of a sense of values" (ibid., p. 15). The first steps toward this reconciliation are to *suspend* the merely instrumental attitude toward technical beings, and to look seriously at the genesis of technical beings in terms of their own intrinsic necessity.

Some scholars have claimed that Simondon's approach has its blind spots. Daniela Voss (2019, pp. 292-296) argues that Simondon has overlooked the way capital and institutional powers shape the form of technical developments. In contemporary cognitive capitalism, new forms of exploitations are often disguised as technical innovations. Without properly analyzing the social and economic dimensions of technical developments, interweaving human life with networked informational devices can develop into new forms of oppressions and alienations. In his analysis of the concretization of social software platforms, Simon Mills (2011, pp. 215-216) argues that "associated milieu [of a social software platform] that is invented and maintained is constructed in association with the regime of the psycho-social and not just that of the physical". To understand digital technicity, we still need to take into consideration social and economic factors as well. As Mills (2011, pp. 224-225) illustrates in his case studies of Twitter and electronic exchange market system, we can further develop Simondon's concretization theory by expanding the concept of associated milieu to social environments. Such inclusion does not contradict his philosophy of nature. After all, Simondon's concept of 'nature' includes all three regimes of individuation: the physical, the vital, and also the psychosocial.

We recognize the strengths but also the historical limitations of Simondon's approach to technical evolutions. We argue that with a broadened concept of associated milieu, along with considerations of the social and economic factors in technical evolutions, concretization theory can still shed light on the nature of contemporary technologies, and in particular the cybersecurity operations of digital environments.

The Individuation of Software Applications

Simondon's theories of biological and technical individuation illuminate the study of cybersecurity in several ways. First, Simondon's arrangement of technical beings as element/individual/ensemble provides a helpful way to conceptually categorize software applications. In the context of software, a digital technical element is a carrier of the most basic computing operations. It can be an instruction in a program, or a system function used by the program during its runtime execution. They are "pure" and "context-free", and can be integrated into any software system. A digital individual comprises a set of organized elements. The most common example of a digital individual is an individual software application such as a text editor, a web browser, or an email server. A digital ensemble consists of a set of individual software communicating through a network. A web-based medical information system, for example, consists of three individual pieces of software: a web browser enabling user-interactions, a web server for processing users' requests, and a database system for data storage. Our focus in this paper is security at the level of individual software applications. An individual software application is a technical individual in a proper Simondonian sense. It is also the central focus of software security. In most cases, security problems do not result from the faults of digital elements but rather from the ways in which they are integrated within the digital individual of a piece of software. At the level of digital ensembles, one vulnerable software subsystem can lead to the compromise of a whole networked ensemble. This has been demonstrated by malware propagation and the recent rise of software supply chain attacks.[1]

[1] In 2020, for example, a group of hackers gained access to the updating server of network management software firm SolarWind and injected malicious logic into the update package DLL of the network monitoring software product Orion. Since Orion is widely used by multiple key departments of the U.S government, it became a "trojan" for the hackers to further compromise the security of these organizations and perform cyber espionage (Rasner, 2021).

Secondly, Simondon's concepts of biological individuation and the concretization of technical beings are helpful in understanding the evolution of software in response to a cyberattack. Each cyberattack such as a malware attack is an informational event in Simondon's sense. It exploits the incompatibilities among the digital elements which software applications base their runtime execution on, structurally deforming them to the brink of disintegration. Unwittingly, a cyberattack also unleashes a chain of security engineering operations which further concretize the vulnerable software. Software applications must reinvent themselves, structurally and functionally, to resolve these incompatibilities. The concretization of a software application is like a vital individuation in that it is an inherently *relational* being and process. Its individuality depends on its relationship with an associated milieu: a running environment called an operating system. The concretization of a software application is always a co-concretization with its OS. Simondon's concept of associated milieu can be applied to investigate the dynamic, reciprocal relationship between individual software applications and operating systems in the context of cybersecurity.

The paradigm of vital individuation is more important in thinking about software security than other domains of technology. Due to the inherent flux of its social and technical environment, software is in a state of metastability. Its internal functional unity is only provisional and is subject to perturbation caused by cyberattacks. Like living beings, software has no choice but to continue its concretization to maintain its integrated form or face total dissolution in the event of a cyberattack.

Case Study

In this section we will use a famous malware attack case study, "Dynamic Link Library (DLL) injection," to examine the cyberattack's effect on the concretization of software applications and their relationship with the operating system. DLL injection is a form of malware attack in which an attacker inserts malicious code into a running process of a software application. Our goal in this paper is not to propose any new security solution, but rather to use DLL injection as an empirical study to illustrate the relevance and helpful insights from Simondon's theory of individuation.

A Primer on Operating Systems

To understand the mode of existence of software, one cannot evade the topic of operating systems as their technical running environment. In this section, we will briefly review the history of operating systems and explain the basic structure and functions of modern operating systems. The goal is to illustrate how the ideology of naive hylomorphism, the alienation between culture and technicity, and the power of capital has shaped the design of the operating system and the mode of its relationship with application software.

A digital object such as a computer program cannot exist without a milieu in which to operate. It might appear as if one physical milieu is the hardware platform on which an application is executing. However, most programs do not directly interact with hardware, because there is a mediator between software programs and their physical milieu. In the early history of computers, running a program was far more complex and indirect than today. While the first generation of computers such as ENIAC had already outperformed humans in the speed of calculation, it still required human operators to manually input the programs into the electronic computer. The early society of the computing profession very much resembled the naive hylomorphic model of labor conditions in ancient Greek society. A 'rational mastermind,' usually a mathematician or scientist working in defense projects, gave a computing task to 'lowly technical operators' and had them compile the concrete procedure of computing steps to complete the task. At that time, it was women who assumed the role of the technical operators (Light, 1999). Female technicians were simultaneously programmers, hardware operators, and file organizers. By outsourcing the 'dirty' work to the female operators, the 'rational minds' could dedicate themselves to much more 'abstract' thinking tasks without being entangled with the 'messy' physical reality of the electronic hardware. Up until the late 1950s, most of the early pioneers of computer programming were women. It is thus not surprising that female programmers made significant contributions to the invention of what is now known as operating systems – an automated version of their own human labors.[2]

[2] One example of this transition from female computer operators to non-human operating systems is seen in the work of Mary Allen Wilkes. From 1959 to 1963, Wilkes designed and wrote the operating system for LINC (Laboratory Instrument Computer) computers, which is the precursor of modern-day operating systems. (Wilkes, 1970)

Immediately after their concrete inception, the subsequent development of operating systems quickly shifted toward more hierarchical structures. With the rise of the modular software engineering approach and the development of high-level programming languages in late 1960s, there was an increasing demand for operating systems to support the concurrent executions of multiple programs and to allow programmers to choose among different programming languages. Operating system design began to adopt a more reductionist approach by separating its core functionality called OS kernel from the users' own programs and from programming languages. Operating systems thus became generic machines which provided more abstract interfaces to the software developers. Software applications became more abstract and more distant from physical hardware. The introduction of personal computers (PC's) and the internet in the 1980s prompted even more complex and diverse demands from PC consumers and business organizations. These demands drastically increased the burden laid on operating systems (Hansen, 2011). A modern operating system must be able to support a large variety of software applications, ranging from 3D video games to web browsers.

To meet the vast and *incoherent* requirements from application software development, a typical operating system today like Windows provides a set of *application programming interfaces* (APIs) for a software developer to be able to use hardware resources more efficiently (McHoes & Flynn, 2010). Suppose a programmer is developing an application, and a part of its function is to store a piece of data on a computer's storage system. Instead of writing her code to deal with the complexity of a solid-state drive, she can have her application call APIs to create a new file, and then write data to the file. As servants to the application, the APIs will handle all the physical tasks of communicating with and controlling a disk drive. Like their human operator ancestors, an operating system hides the messy materiality of a computer and provides an abstract representation of a physical machine to an application program. For most applications and for ordinary human users today, the operating system is the most immediate milieu. Without its support, a computer is merely a collection of raw physical devices. The relationship between ordinary users (including application software developers), applications, a modern operating system, and hardware is depicted in the following figure:

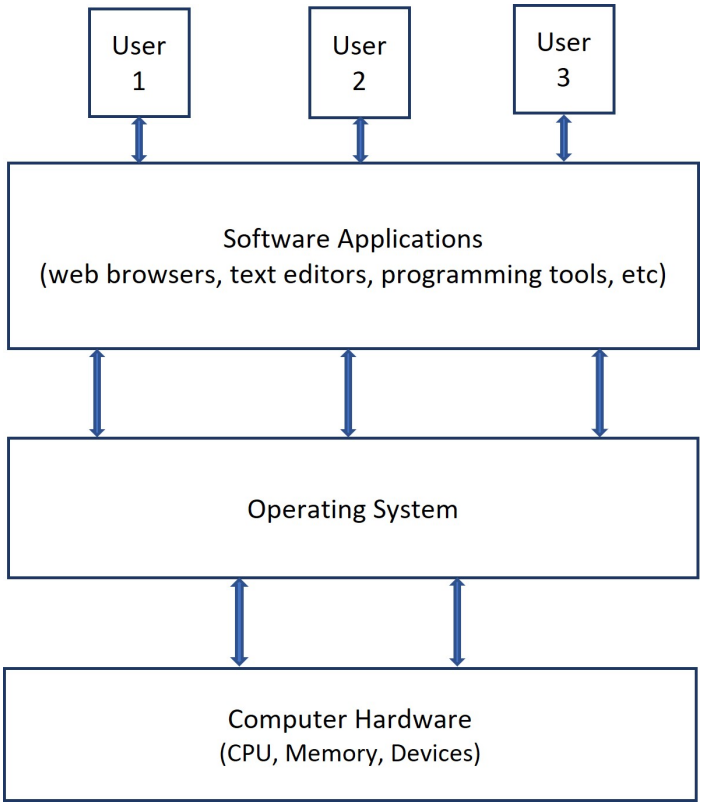

Figure 1. Relationship between users, applications, OS, and hardware.

This allows us to see that modern operating systems have inherited the naive hylomorphic ideology in their very structure. An operating system is a passive '*servant*' which facilitates the execution of a developer's application program without having to know the meaning of the program. When an application program starts running, the operating system will begin a chain of material operations: allocating memory space, loading the application's program code to the memory, preparing the CPU to execute the code, etc. With the operating system's support, software application developers can operate the computer system through APIs without having to know the operations carried out inside the OS. They can dedicate their work on constructing the narrative and meaning at the level of application software – the appearance of user interfaces, the mode of communication among users, etc. The separation between an operating system's internal operations and the application-level software reflects what Simondon refers to as the alienation between culture (apps) and technicity (OS and hardware) in the modern age.

The development of an operating system does not depend exclusively on the hardware and on the minds of the OS engineers. Its development also involves a much larger *social* and *economic* context. Operating systems vendors such as Microsoft or Google have been using APIs as an effective way to attract developers to their platforms. For an operating system, more developers using it means that more software applications are running on the system. This, in turn, means greater market share. In the past decades, there have been API wars between different competing operating system platforms (Spolsky, 2004). How to design new attractive APIs is becoming an integral part of business strategy for many software platforms vendors (Jacobson et al., 2012). Each year, hundreds of conferences are held to attract software developers to learn newly released APIs by various vendors. Communities of developers are built around sets of APIs provided by different platforms. With the enlargement of the developer community for a given operating system, there are also increasing demands to further expand its APIs. An operating system is always incomplete. An operating system and its developers constitute a vibrant, open techno-social system.

As a consequence of these dynamic social interactions, the operating system has become increasingly complex over the last two decades. In each new version of the Windows operating system, there are approximately 1,000 new system APIs added. In the year 2003, Windows XP had little over 2,000 APIs. By the year 2020, Windows 10 has more than 10,000 APIs! In addition to the growth of the API population, modern operating systems also encourage code sharing and modularization. Through the mediation of the operating system, a software application can even share its code with other applications. In Windows, this is done via the use of dynamically linkable library (DLL). A DLL is a program that can be shared by multiple running applications at the same time. From the perspective of software developers, re-using DLLs reduces the programming effort to develop an application. Instead of programming everything from scratch, a programmer can have an application load a DLL during the runtime. For example, when a drawing application needs to print an image during its execution, it can simply request a Windows API, LoadLibrary, to load a DLL specialized for printing and use the program contained in the DLL to complete the task. The same DLL can be shared by various software applications which need to print images. And

one running application on Windows typically loads multiple DLLs during its execution.

Rich APIs and DLLs provide software application developers with greater coding productivity. They are also a means to *control* the community of developers. APIs are the primary means to regulate and settle different demands from developers, and they reflect the will of the majority in the community. However, the techno-social system built around Windows APIs can never reach a complete, stable closure. Due to the incoherent demands from developers, the functions of APIs are often incompatible with one another. The disparities among APIs leave the Windows system in an unstable state. It is in this context of instability that hackers *invent* creative ways of linking Windows' system features and turn it into a dangerous milieu for applications. In the next section, we will visit a classic example of a malware infection technique to examine the relationship between software applications and their milieu. We will show that hackers defy the hylomorphic techno-social order, travelling across the border between application level and operating system level. A cyberattack event is a germ of change. It triggers a chain of operations to secure the system. It brings a whole community of developers and system engineers together to solve a security problem, and their collaborative engineering effort radically restructures an operating system and its relationship with software applications. As a result of this restructuring, a software application integrates a part of the operating system within itself to fend off malware infection, essentially creating a tighter association with the operating system. Cyberthreats and security engineering responses begin to erode the *wall* between software applications and operating systems, thus concretizing both of them, and making their relationship partially resemble the one between living beings and their associated milieus. As we will suggest below, software engineers can cultivate software and its environment in such a way as to increase this resemblance to living beings, and thus increase the level of security.

Study of a Malware Infection Attack

The running activity of an application's code is called a "process" (McHoes & Ballew, 2012). With the support of an operating system, one machine might have multiple

processes running simultaneously. Typical running processes in a Windows machine are file explorer, web browsers, etc. Nevertheless, an operating system as a milieu does not sufficiently protect the individual running process of an application from the harmful behavior of malware. Stealth malware often employs a technique called 'process injection' to inject malicious code into a running process (Monnappa, 2018). [3]

Once it lands in a target Windows machine, malware can use legitimate APIs provided by Windows to inject malicious code into the memory space of a running application and to execute that code. Since the injected malicious code runs in the memory space of a benign application such as Notepad or the Firefox web browser, it can evade anti-malware detection systems. One of the most popular and straightforward types of process injections is DLL injection. The malware first drops a malicious DLL file into the victim's file system. Then, it uses the following four steps to complete DLL injection:

> Step 1: The malware uses a Windows API, OpenProcess, to attach to the running process of a victim application, such as Notepad. The OpenProcess API is by no means a malicious function. It was intended to facilitate system administration. A system administration tool often uses this API to monitor all the running processes in a windows system, profiling their performance and resource utilization.
>
> Step 2: The malware uses another Windows API, VirtualAllocEx, to allocate a piece of memory within the victim process. The malware will then issue WriteProcessMemory to store the path of the DLL to that memory location.
>
> Step 3: The malware uses a Windows API, LoadLibrary, to load the malicious code contained in the DLL file into the memory space of the victim process.
>
> Step 4: Once in the victim process, the malicious code will carry out malicious operations. In some advanced malware attacks, the injected code will delete the malware and the original copy of the malicious DLL file. The whole procedure of DLL injection is depicted in Figure 2.

[3] Process injection attacks have been reported in all three major operating systems -- Windows, Linux, and macOS. We will focus on Windows in this article.

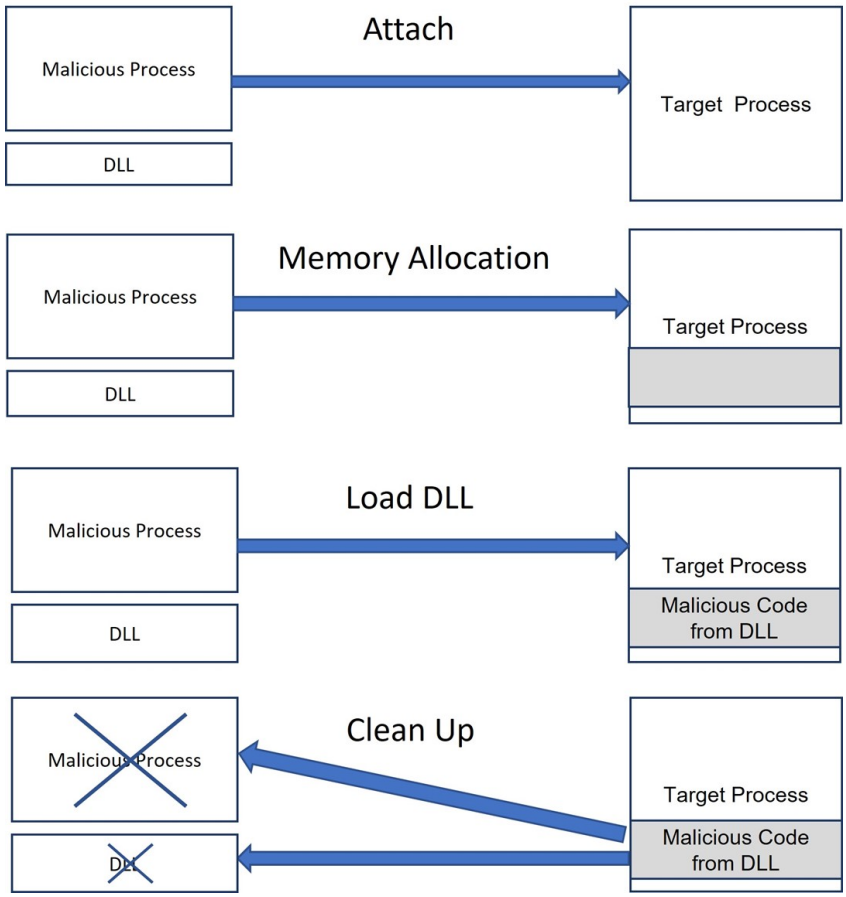

Figure 2. The steps in DLL injection procedure.

Like OpenProcess API, neither VirtualAllocEx nor LoadLibrary APIs are malicious, nor are they insecure by themselves. They are all created as the operating system's routine services to help software developers express their ideas and translate them into material reality at the hardware level! Any software application can use VirtualAllocEx to allocate the memory space that it needs during the execution. Any running software application can use LoadLibrary API to load a DLL that it needs. It is the incompatibilities among them as technical elements and the way in which they are exploited by the malware in its interaction with the target application that cause security problems.

Software Security through the Lens of Simondon

The interaction between a malware and its target program cannot be explained by the early cybernetic notion of information as the transmission of encoded messages from sender (malware) to receiver (target application). From the perspective of this impoverished notion of information, there is no difference between the scenario where two benign software applications are exchanging data and the one where a malware is infecting a target program. Simondon's view of information is that it is a form-taking operation which emerges between two disparate systems, and this view offers a better model to understand the mutual restructuring and deformation between a malware and its victim during the infection.

As the result of the deformation due to malware infection, the victim process loses its concreteness. For example, once the injected malicious code starts running in the memory space of a victim application, it can perform many different types of covert activities. It can secretly collect all the users' bank account data stored in the same operating system and send them to the hacker's web server via a covert network connection. Some malware can even use the infected software application as a host to further propagate the malware itself to other applications in the same computer system or even other computer systems through network connections. In any case of malware infection, injected code lives as a parasite within the process of the original application, causing the original application to lose its functional integration.

To thwart malware attacks, cybersecurity engineers must find ways to protect the functional integration of the software applications running in an operating system. Microsoft, for instance, has created what is called the AppLocker subsystem since Windows 7 (Corio, 2009). With the AppLocker, a software application developer can pre-program a security rule in the code of her application which tells Windows a list of DLLs that the software application is expected to use. When the application begins its execution, its pre-programmed security rule will be sent to the Windows operating system. The Windows operating system, in turn, will use the rule to monitor each DLL the application is attempting to load during the process of its execution. If the application is trying to load a DLL that is not specified in the list, this indicates that the application has been injected by malicious code, and begins to perform a computing task unintended by the developer. For example, if a simple application such as

Notepad is trying to load a DLL for network communication, it is highly suspicious. The system thus detects a potential process injection attack and blocks the execution.

Through the lens of Simondon's theory of concretization, the above cybersecurity engineering method can be seen as a process of concretization. The software application that was previously vulnerable to the disintegrating power of malware gains a greater level of functional integration. This concretization is achieved by inventing a greater reciprocal relationship between a software application and its milieu. Now, the execution of an application reconfigures parts of the OS's functionality according to specified security rules. The reconfigured OS, in turn, safeguards the execution of the application. The operating system is transformed from a generic platform which blindly accepts any application into a more concrete milieu that is more tightly coupled with specific applications. Unlike other domains of technology, software applications run in an inherently metastable milieu. With the ever-increasing expansion of APIs in Windows, one can only anticipate more sophisticated malware attacks will emerge in the future, exploiting the incompatibilities among new and old APIs. For software applications, continuing the concretization is a matter of 'life-or-death'. This dimension of 'existential struggle' makes the concretization in the context of software security closer to Simondon's concept of vital individuation. A software application must perpetually re-invent its interior and re-define its relationship with the milieu in the process of resolving the disparities that are causing vulnerabilities. With the increased association with its OS, the software application gains greater functional unity and is better able to survive malware attacks. In other words, a software application paradoxically maintains its individuality not by being closed up within itself, but by being open to the broader milieu and by participating in the concretization of the whole collective of software, similar to transindividuation in the psychic-social domain.

Conclusion

As this article has shown, a deep understanding of cybersecurity attacks and solutions must rest upon a sufficiently sophisticated general theory of being. More specifically,

such an understanding must rest upon an ontology that is complex enough to account for processes of change that involve reciprocally causal elements and dynamically shifting relations. We believe that Simondon's general theory of ontogenesis is a powerful starting point for such an ontology. His theory not only provides a nuanced account of how technical objects develop over time, but it also proposes the bold idea that, as some technical beings mature, they begin to resemble biological beings in certain important ways. Rejecting the common hylomorphic ontology which reduces entities to abstract form/matter relations, he works to establish an ontology that can account for beings that become more concrete over time by means of their own internal resources and also by means of their reactions to the environment that they themselves are altering. In so doing, Simondon better explains the active dimensions of materiality, the recurrent causality between object and environment, and the dynamic relations of operations and the structures in which those operations act. This general ontology allows Simondon to develop a theory of technical objects which clarifies important aspects of cybersecurity threats and solutions that other theories overlook, such as how a cyberattack (reminiscent of threats to biological beings) can be the occasion for a software application to become more integrated and more secure.

In addition to the engineering insights, the study of cybersecurity can benefit from a new norm of thinking and practicing technology which Simondon (2009b) refers to as the *technical mentality*. This emerging technical mentality has a single criterion: "that of the opening" (Simondon & De Boever, 2009b, p. 24). Secure software development needs to move beyond the paradigm of the automata, the self-enclosed cybernetic "blackbox", and embrace what Simondon describes as *open objects*. An open object is always in progress. It is open to its milieu and to further perfection, like a growing organism. (Simondon, 2014, p. 401). An open digital object concretizes relationally, by penetrating into its milieu like a plant stretching its roots.

Future researchers can build upon this essay, in combination with other research, in several ways. Yuk Hui's *Existence of Digital Objects* deals with the individuation of structured data and metadata (Hui, 2016), while our essay focuses on software processes; an account is needed that would encompass data and software into a larger holistic picture specifically in terms of cybersecurity. Simon Mills' study on the

co-evolution of social software platforms with the psychosocial domain of their users (Mills, 2011, 2016) can also benefit from our study in terms of social software security. With the proliferation of APIs, the social software platforms are becoming unstable and will certainly face more security problems in the future. Our study can provide insights on how cyberattacks affect the individuation of the platforms in relation to both users and application developers.

Bibliography

Barad, Karen (2003). Posthumanist performativity: Toward an understanding of how matter comes to matter. Signs: Journal of Women in Culture and Society, 28(3), 801–831. https://doi.org/10.1086/345321

Bardin, Andrea (2015). reforming the concepts of form and information. In Epistemology and political philosophy in Gilbert Simondon individuation, Technics, social systems (pp. 21–34). essay, Springer Netherlands.

Barthélémy, Jean-Hugues & Norman, Barnaby (2015). Aspects of a Philosophy of the Living. In Life and technology: An inquiry into and beyond simondon (pp. 15–20). essay, Meson Press.

Combes, Muriel (2013). In T. LaMarre (Trans.), Gilbert Simondon and the philosophy of the transindividual (pp. 3–4). The MIT press.

Corio, Chris (2009, May). An introduction to security in Windows 7. TechNet Magazine, 13–20.

Hansen, Per B. (2011). The Evolution of Operating Systems. In Classic operating systems: From batch processing to distributed systems (pp. 1–34). Springer New York.

Hui, Yuk (2016). On the existence of digital objects. University of Minnesota Press.

Jacobson, Daniel, Woods, Dan & Brail, Greg (2012). Apis: A strategy guide. O'Reilly.

Light, Jennifer S. (1999). When computers were women. Technology and Culture, 40(3), 455–483. https://doi.org/10.1353/tech.1999.0128

McHoes, Ann M. & Ballew, Joli (2012). process and threat management. In Operating systems demystified (pp. 78–102). McGraw-Hill.

McHoes, Ann M. & Flynn, Ida M. (2010). Windows Operating System. In Understanding Operating Systems (sixth edition, pp. 464–494). South-Western.

Mills, Simon. (2011). Concrete Software: Simondon's mechanology and the techno-social. Fibreculture Journal, (18).

Mills, Simon. (2016). Toward a Theory of Media. In Gilbert Simondon: Information, Technology and Media (pp. 173–206). essay, Rowman & Littlefield.

Monnappa, K A (2018). Code Injection and Hooking. In Learning malware analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows Malware (pp. 283–327). Packt Publishing Ltd.

Rasner, Gregory C. (2021). Cybersecurity and third-party risk: Third party threat hunting. Wiley.

Simondon, Gilbert. (2009a). The position of the problem of ontogenesis . Parrhesiajournal, (7), 4–16.

Simondon, Gilbert (2009b). Technical Mentality. Parrhesiajournal, (7), 17–27.

Simondon, Gilbert (2014). Sur La Technique: (1953-1983). Presses universitaires de France.

Simondon, Gilbert (2017). On the mode of existence of technical objects. (Malaspina Cécile & J. Rogove, Trans.). Univocal Publishing.

Simondon, Gilbert (2020). Individuation in light of notions of form and information. (T. Adkins, Trans.). University of Minnesota Press.

Spolsky, Joel (2004). How Microsoft lost the API War. Joel on Software, 295–312. https://doi.org/10.1007/978-1-4302-0753-5_42

Voss, Daniela (2019). Invention and capture: A Critique of simondon. Culture, Theory and Critique, 60(3-4), 279–299. https://doi.org/10.1080/14735784.2019.1679652

Wilkes, Mary A. (1970). Conversational access to a 2048-word machine. Communications of the ACM, 13(7), 407–414. https://doi.org/10.1145/362686.362690

Author Information

Ziyuan Meng (zmeng@drew.edu)

Ziyuan Meng is Assistant Professor of Computer Science at Drew University, New Jersey, United States. Since 2014, Ziyuan Meng has helped multiple institutes to develop cybersecurity curriculum with emphasis on an interdisciplinary approach. His main research focus is to bring insights from continental philosophy and constructivist psychology to develop a non-reductionist approach to cybersecurity.

Jon K. Burmeister (jon.burmeister@mountsaintvincent.edu)

Jon K. Burmeister is Assistant Professor of Philosophy at the College of Mount Saint Vincent, in New York City. His dissertation was on Hegel's idea of a 'living' logic and what this entails for philosophical language. His research combines a background in the history of philosophy with investigations of emerging digital technologies and their social and ethical impacts. Jon K. Burmeister was the recipient of a 2016 - 2017 National Endowment for the Humanities "Enduring Questions" Grant on the subject of Work and Leisure.