Emergent Behavior in Cybersecurity

Shouhuai Xu
Department of Computer Science, University of Texas at San Antonio

ABSTRACT

We argue that *emergent behavior* is inherent to cybersecurity.

Keywords

Emergent behavior, cybersecurity, security properties

1. INTRODUCTION

The human-created cyberspace is a very large-scale complex system of cybersystems. Its security properties are difficult to understand and characterize. We attribute this difficulty to its *complexity*, a manifestation of which is the so-called *emergent behavior* in Complexity Science [6]. Although there is no universally accepted definition of emergent behavior [11], the basic idea is intuitive. The simplest example of emergent behavior may be the well known “$1 + 1 > 2$" effect. For our purpose, it is sufficient to use the following informal definition of emergent behavior in cybersecurity domain.

DEFINITION 1. *A security property of a cybersystem exhibits emergent behavior if the property is* not *possessed by the underlying lower-level components of the cybersystem.*

A direct consequence of emergent behavior is that at least some security properties cannot be understood by solely considering the lower-level components; instead, we must explicitly consider the *interactions* between the lower-level components. Although emergent behavior of cybersystems has been discussed from a function or constructional perspective [10, 7], emergent behavior in cybersecurity is not systematically examined until now.

2. EMERGENT BEHAVIOR

We demonstrate emergent behavior in cybersecurity through three examples.

Emergent behavior exhibited by cybersecurity dynamics. We refer to [15] for an exposition of the emerging field of cybersecurity dynamics. In order to explain how emergent behavior is exhibited by cybersecurity dynamics, we consider the perhaps simplest model [4]. For our purpose, it suffices to consider two cybersystems that respectively induce attack-defense structures (or graphs) $G_i = (V_i, E_i)$, where V_i is the node (vertex) set and E_i is the edge set for $i = 1, 2$. Let $\lambda_1(G)$ denote the largest eigenvalue of the adjacency matrix of a graph G, β denote the defense capability in detecting and cleaning compromised nodes (e.g., the probability that a compromised node gets cleaned at each time step), and γ denote the attack capability in compromising secure nodes (e.g., the probability that this event occurs over an edge at each time step). For simplicity, suppose G_i is a complete graph with n_i nodes for $i = 1, 2$. It is well known that $\lambda_1(G_1) = n_1 - 1$ and $\lambda_1(G_2) = n_2 - 1$. For $i = 1, 2$, if $\lambda_1(G_i) < \beta/\gamma$, the attacks will eventually be wiped out in the cybersystem that induces G_i; if $\lambda_1(G_i) > \beta/\gamma$, the attacks cannot be wiped out [4].

Now we consider a new cybersystem that is obtained by interconnecting the aforementioned two cybersystems that induced G_1 and G_2. Consider the simplest case that any node can attack any other node in the interconnected cybersystem, which effectively induces a complete graph $G_{1,2}$ with $n_1 + n_2$ nodes and $\lambda_1(G_{1,2}) = n_1 + n_2 - 1$. In many (if not all) cases, the defense capability β' and the attack capability γ' associated to $G_{1,2}$ are respectively the same as the defense capability β and the attack capability γ associated to G_1 and G_2. Since $\lambda_1(G_i) < \beta/\gamma$ for $i = 1, 2$ do not imply $\lambda_1(G_{1,2}) < \beta'/\gamma' = \beta/\gamma$, we conclude that the attacks can be wiped out in the two underlying component cybersystems, but cannot be wiped out in the interconnected cybersystem as long as $\lambda_1(G_{1,2}) > \beta/\gamma$. This phenomenon can be naturally extended to more sophisticated settings (e.g., [16]). This implies that cybersecurity dynamics cannot be determined by looking at the component cybersystems alone. Rather, we need to look into how the component cybersystems interact with each other.

Emergent behavior exhibited by security properties in the extended trace-property framework. In the field of program verification, it was known that specifications that are sufficient for *sequential* programs are not sufficient for *concurrent* programs. For dealing with concurrent programs, Lamport proposed the safety-liveness framework of trace properties [12]. Intuitively, a *trace* is a finite or infinite sequence of states corresponding to an execution of a program. A *trace property* is a set of traces such that every trace, *in isolation*, satisfies the same predicate. A *safety* property says that no “bad thing" happens during the course of a program execution, while a *liveness* property says that “good thing" will eventually hap-

HOTSOS ’14 USA

pen during the course of a program execution. Both safety and liveness are trace properties. A beautiful result is that every trace property is the intersection of a safety property and a liveness property [12, 1].

Given the above history, it is appealing to specify a cybersystem as a set of traces, and therefore as a subset of a security property that is also specified as *a set of traces*. Unfortunately, security properties are not trace properties as shown in [8, 5, 14] and refreshed below. First, *noninterference* is a security property that captures the intuition that system security is preserved as long as the high-clearance (or high-privilege) processes cannot influence the behaviors of the low-clearance (low-privilege) processes. It is no trace property because it cannot be verified without examining the other traces. Second, *information-flow* captures some kind of correlation between the values of variables in multiple traces. It is no trace property because it cannot be verified by examining each trace alone. Third, *average service response time* is an availability property. It is no trace property because it depends on the response time in all traces.

In an effort to overcome the above limitation of the safety-liveness framework, Clarkson and Schenider extended the notion of *trace properties* to the notion of *trace hyperproperties* [5]. Basically, hyperproperties are *sets of trace properties*. In parallel to the safety-liveness framework, a hyperproperty is also the intersection of a safety hyperproperty and a liveness hyperproperty. It is now known that information-flow, integrity and availability can be hypersafety or hyperliveness [5]. Exactly because hyperproperties capture that the verification procedure *must* examine across *multiple* traces, which may accommodate interactions between the component systems, we say that hyperproperties exhibit the emergent behavior. This means that we need to study the emergent behavior in cybersecurity, which may explain why it took so long to realize the importance of hyperproperties.

Emergent behavior exhibited by cryptographic security properties. Cryptographic secure multiparty computation allows multiple parties $P_1, \ldots, P_m$, each having a respective secret $x_1, \ldots, x_m$, to compute a function $f(x_1, \ldots, x_m)$ such that no information about the x_i's is leaked except for what is implied by the output of the function. This manifests a confidentiality property. A beautiful feasibility result is that any polynomial-time computable function $f(\cdot, \ldots, \cdot)$ can be securely computed [17, 9], as long as the protocol executes *in isolation* (the stand-alone setting) and trapdoor permutations exist. When such cryptographic protocols are used as building-blocks in larger applications/systems, they may execute concurrently (rather than in isolation). This leads to a natural question: Are the cryptographic protocols, which are provably secure when executed in isolation, still secure when they are concurrently called by larger applications/systems? Intuitively, concurrent executions offer the attacker the leverage (for example) to schedule the messages in a way that is to the attacker's advantage, which does not have a counterpart in the stand-alone setting.

Quite similar to what happened in the field of program verification, where specific properties (e.g., partial correctness and mutual exclusion) were investigated before the introduction of the unifying safety-liveness framework [12], the same kind of development was made in the field of cryptographic protocols. That is, specific cryptographic security properties were investigated before the introduction of the unifying notion called *universal composability* [2], or its equivalent (but perhaps more intuitive) version called *concurrent general composition* (arbitrarily many instances run possibly together with arbitrary other protocols) [13]. It is now known that there are cryptographic multiparty computation protocols, which are provably secure when executed in isolation, but are *not* secure when they are concurrently called by larger applications/systems. For example, there exist classes of functions that cannot be computed in the universally composably secure fashion [3]. In other words, these functions can be securely computed by running some cryptographic protocols in isolation, but cannot be securely computed when the protocols execute concurrently. In order to make cryptographic multiparty computation protocols secure when they are used as building-blocks for constructing larger cybersystems, we need to make extra assumptions, such as that majority of the parties $P_1, \ldots, P_m$ are not compromised [2]. This manifests emergent behavior. (It is interesting to note that whether or not it is reasonable to assume that majority of the parties are not compromised may be addressed by the cybersecurity dynamics framework [15].)

Acknowledgement. This work was supported in part by AFOSR Grant # FA9550-09-1-0165 and ARO Grant # W911NF-13-1-0370 .

3. REFERENCES

[1] B. Alpern and F. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985.

[2] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. FOCS'01*, pp 136–145.

[3] R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In EUROCRYPT'03, pp 68–86.

[4] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur.*, 10(4):1–26, 2008.

[5] M. Clarkson and F. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[6] P. Erdi. *Complexity Explained*. Springer, 2008.

[7] V. Gligor. Security of emergent properties in ad-hoc networks. In *Proc. Security Protocols Workshop'04*, pp 256–266.

[8] J. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symp. on Security & Privacy'82*, pp 11–20.

[9] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. ACM STOC'87*, pp 218–229.

[10] H. Hinton. Under-specification, composition and emergent properties. In *Proc. NSPW'97*, pp. 83–93.

[11] A. Kubík. Toward a formalization of emergence. *Artif. Life*, 9(1):41–65, 2002.

[12] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[13] Y. Lindell. General composition and universal composability in secure multi-party computation. In *FOCS'03*, pp 394–403.

[14] F. Schneider. Beyond traces and independence. In *Dependable and Historic Computing*, pp 479–485, 2011.

[15] S. Xu. Cybersecurity dynamics. In *Poster submission to HotSOS'14*.

[16] S. Xu, W. Lu, and L. Xu. Push- and pull-based epidemic spreading in arbitrary networks: Thresholds and deeper insights. *ACM TAAS*, 7(3):32:1–32:26, 2012.

[17] A. C. Yao. How to generate and exchange secrets. In *Proc. FOCS'86*, pp 162–167.