12-15-2019

Reusability or disposability? A network analysis of reusable software vulnerabilities

Martin Kang

Anat Hovav

Sungyong Um

Ted Lee

Reusability or Disposability? A Network Analysis of Reusable Software Vulnerabilities

Martin Kang[1]
College of Business, Mississippi State University,
Mississippi, MS, USA
Anat Hovav
Claremont Graduate University
Claremont, CA, USA
Sungyong Um
School of Computing, National University of Singapore,
Singapore, Singapore
Ted Lee
Fogelman College of Business, University of Memphis,
Memphis, TN, USA

ABSTRACT

The sharing economics of digital resources such as application programming interface and software development kits enables software developers to create diverse software faster and more effectively. However, developers may unintentionally adopt shared digital resources that contain vulnerable code, which may cause vulnerability in the final software product. In this study, we examine the interplay between the sharing of digital resources and software vulnerability. This study may help developers to identify vulnerable digital resources in a digital ecosystem. We quantify the diffusion of software vulnerability in sharing of digital resources using a machine learning technique to create necessary variables. We conduct regression analyses to examine their effect on the severity of software vulnerabilities.

Keywords: Software Vulnerability, Network Analysis, Business Analytics, Node2Vec, Machine Learning, Econometrics.

INTRODUCTION

Deliberate attempts to cause failure by triggering information security (InfoSec) incidents is what software management is concerned with and has to protect against. Software vulnerability is a flaw within a software product that can cause it to work contrary to its intended

[1] Corresponding author. dk1029@msstate.edu +760 815 4846

design and can be exploited to cause the system to violate its documented security policy (Applewhite 2004). The economic impact of software vulnerability can be devastating. For example, Wells Fargo and eleven other financial institutions were fined $14.4 million for failing to adequately protect their electronic records from software vulnerability (Rubin and Pollet 2017). According to Phonemom (2017), US companies spend an average of $3.8 million each year to fix software vulnerabilities. Across all stakeholders including customers, $8 billion is spent to prevent potential damage to computer systems from software vulnerabilities in the US per year.

A vast body of literature attempts to provide business and technical guidance to mitigate software vulnerability by discovering the causes that generate it. Conventional software vulnerability studies focus on detecting vulnerability at the design level of software. These studies advocate that software manufacturers and developers can mitigate vulnerability by eliminating potential errors in the software development phase. For instance, Rehman and Mustafa (2009) and Tevis and Hamilton (2004) find a positive association between the number of software vulnerability occurrences and the transitional phases of the software design process. Walden et al. (2014) introduce a machine learning method to detect vulnerable codes in the software development process. Luo et al. (2014) develop metric models that evaluate the severity of software vulnerability by using the mathematical characteristics of software vulnerabilities.

However, modern software development is a complex jigsaw puzzle of internal and external digital resources such as application programming interfaces (API) and software development kits (SDK). The decision to adopt external digital resources in the digital ecosystem should entail a rigorous process to evaluate the potential vulnerability of these resources. For instance, the Equifax data breach in 2017, which leaked the information of 45.5 million U.S.

consumers, was facilitated by a flaw in Apache Struts (NVD: CVE-2017-5638). Apache Struts is an open source platform that provides APIs for web application development. Check Point estimated that CVE-2017-5638 affected 42% of web applications due to its vast externality and popularity.

Despite the complex interplay of digital resources in modern software development, there is surprisingly scant quantitative research on the association between external digital resources and software vulnerability. This paper studies the relationships between the digital resources used to develop software and their consequence for software vulnerability. Studying this relationship is vital because we still do not understand many aspects of how the digital resources in a digital ecosystem affect software vulnerability despite the wealth of research on software vulnerability detection. Our empirical research is guided by the following research questions: what factors of the digital ecosystem can impact software vulnerability, and how do they impact the severity of software vulnerability?

To answer these research questions, we begin with a review of related literature on software vulnerability and digital ecosystems and derive propositions based on the findings in the literature. Next, we assume that software vulnerability could be diffused by sharing digital resources in a digital ecosystem. We infer the network structure of software vulnerabilities by using evidence about the sharing of digital resources in a digital ecosystem taken from the software vulnerability data in the National Vulnerability Database (NVD). In our dataset, we define a node as a software vulnerability. The edge of the node is its relationship with other software vulnerabilities. We assume that edges exist when the software vulnerabilities share the same affected product. Based on the dataset, we introduce and conduct various network analyses to derive the exploratory and control variables that might affect the severity of the software

vulnerability. Finally, we validate our assumptions using a regression analysis based on the variables derived in the network analysis.

RELATED WORK

Research related to the analysis of software vulnerability can be categorized into two different strands based on its perspective on software vulnerability. The first strand focuses on developing methods to find software vulnerability by excavating the pattern of programming structure. Shahriar and Zulkernine (2012) categorize static, dynamic and hybrid analysis for software vulnerability analysis. Static analysis refers to analyzing the source code of software without considering the executing process (Medeiros et al. 2015). Dynamic analysis refers to continuous analysis using specific input data for software functional testing and general operations of software (Moore et al. 2016). Hybrid analysis is a mixture of these processes (Watson et al. 2015). The second strand of research mainly discusses the methods to discover software vulnerability using various metrics, data mining and examining the programming errors that can be generated by human error and machine malfunctions.

The second strand mainly examines the relationships between software vulnerability and software architectural properties. For instance, Sosa et al. (2013) consider the cyclicality of digital resources (e.g., API and SDK) in digital ecosystems, using the design structure model for digital resources to first estimate the cyclicality of software dependencies and then assess its positive association with a software vulnerability. Lagerström et al. (2017) combined detection metrics (e.g., quantitative metrics in the first strand), properties of code (e.g., findings in the second strand) and architectural aspects to predict software vulnerabilities. The software engineering literature emphasizes software design algorithms and procedural concerns to detect vulnerable software components from external digital resources (Wang et al. 2010).

RESEARCH PROPOSITIONS

Diffusion of Software Vulnerability

Software can be linked to other software by sharing digital resources such as API and SDK. However, the shared digital resource could have a latent vulnerability that affects other software functions, triggers attacks and acts as a source of threat. The literature on digital platforms and ecosystems provides empirical evidence for this argument. Digital products have a layered modular architecture in which digital resources and services are combined to create final products and services (Yoo et al. 2010). The digital resources shared by external parties in the digital platform allow for sustainable growth by structural diversification of digital ecosystems (Um and Yoo 2016). Barros and Dumas (2006) examined the diffusion of API in the digital ecosystem of a web service and explored how developers consume the shared API to create new and innovative functions for their products.

The literature on software vulnerability often utilizes the relationship between the software vulnerability and the digital ecosystem. For instance, some studies estimated the likelihood of vulnerability based on the complexity of the code and data structure (Gopalakrishna et al. 2005). More recent studies use a software dependency network based on the mutual function calls among the entities in a related digital ecosystem (Nguyen and Tran 2010). In addition, some research focuses on modeling the vulnerability discovery process based on the shared economics of digital resources (Alhazmi et al. 2007) and entities of software vulnerability ecosystems using an agent-based model (Breukers 2016).

A vast body of studies shows that the externalities of digital resources have virtuous network effects on a digital ecosystem because externality may provide functional recombination and diversification of software. However, the structure of the digital ecosystem may affect the

diffusion of software vulnerabilities by allowing the sharing of vulnerable digital resources in the ecosystem. The more developers use the digital ecosystem to develop innovative functions for their software, the more likely they are to create vulnerable software by accidentally using external digital resources that may contain vulnerable codes. Furthermore, hackers could take advantage of external vulnerable codes to attack the target systems. Thus, the impact of software vulnerability could be amplified as the vulnerable digital resources are assembled to create a final software product. Hence:

Proposition 1. The diffusion of software vulnerability in a digital ecosystem is associated with the severity of software vulnerability.

Digital Ecosystem Structure of Software Vulnerability

The digital resources in digital ecosystems tend to be very large and to contain elements from various other digital resources. As a result, the management of digital resources has become more difficult (Decan et al. 2019). These dependencies between digital resources cause various types of challenges in software vulnerability and security due to inconsistencies in version management, high consumption of computing power for cascading version updates and lack of awareness of developers and digital platform owners regarding the complex structure of these dependencies (Brada and Jezek 2015; Kula et al. 2018). Vulnerable code could exist in outdated dependencies of digital resources and result in vulnerable software that could be used as a digital resource in other software products and thus propagate throughout the ecosystem (Nguyen and Tran 2010; Rahimi and Zargham 2013).

This means that popular digital resources that are widely used in a digital ecosystem may create multiple channels through which software vulnerability can be spread. The severity of software vulnerability could be shaped differently by software dependency as the vulnerable

codes are delivered, used and transformed by other digital resources. For instance, Equifax's incident featured a few stages that led to the breach. First, the attacker attacked the vulnerable Apache Struts API, which enables communication between heterogeneous programming objects (e.g., serialized programming objects, link files and data components) and used it to browse and explore the target digital assets in Equifax's systems. Then, the attacker used another software vulnerability of Apache Struts to disturb the network segmentation of the digital infrastructure by merging other APIs (i.e., embedded in Equifax's systems) related to data injection. Notably, those APIs were not designed to be used for browsing or interfering with the communication between the programming objects. The hacker creatively devised a way of attack by taking advantage of a malicious aspect of the digital ecosystem. As such, attackers can amplify the severity of software vulnerability by merging related APIs and tweaking their functions.

This suggests that software vulnerability could be mediated and propagated. Even if the software vulnerabilities are not linked directly to each other, one software vulnerability could trigger the operation of another vulnerable code, thus creating a domino effect. In addition, an attacker can devise new malicious functions using related digital resources from neighboring software vulnerabilities. Understanding these impacts on the severity of software vulnerability requires network analysis that examines the role and centrality of software vulnerabilities in a digital ecosystem.

The centrality of software vulnerability in the digital ecosystem could impact the severity of software vulnerability in different ways. For instance, a software vulnerability (e.g., degree centrality) that is more directly related with other software vulnerabilities may be more severe because it impacts the local area of the network. The global network relationships such as the density of the links between software vulnerabilities (e.g., eigenvector and clustering coefficient)

in a certain part of the network could also impact the severity of software vulnerability. However, the network centralities could be measured differently by considering the position of software vulnerability in a global software vulnerability network. For instance, software vulnerabilities in the central position of a network could gain more attention from developers and more externalities in a digital ecosystem. The centrality of software vulnerability could motivate communities and stakeholders to control the risk attributed to the software vulnerability. Using the metaphor of fire control, the firefighter controls the main fire at the site and then attends to distant fires. The resources in the digital ecosystem could be focused on the software vulnerability that is central in the digital ecosystem, and this could lead to less software vulnerability.

We attempt to find the effect of this ambiguous concern on network centrality by testing a proposition using various network centrality measures. These measures are introduced in the section on the empirical approach. In order to examine the impact of the various network centralities of software vulnerability in the digital ecosystem, we offer a general proposition (P2) as follows:

Proposition 2: The network centralities of the software vulnerability in the digital ecosystem are associated with the severity of software vulnerability.

METHOD

To test the propositions, we first collected software vulnerability data through the National Vulnerability Database (NVD). The NVD includes databases of security checklist references, security-related software flaws, misconfigurations, product names and impact metrics. The various characteristics of software vulnerability reported to the NVD are the date of report, solutions from the open source community and digital artifacts affected by the software

vulnerability. Based on the data, we infer the software vulnerability network as G. Let G = (V, E) be a software vulnerability network. V is a node (i.e., software vulnerability) and E is an edge which represents the relationships between software vulnerabilities. We consider an E to exist between Vs when two different Vs affect the same software.

Second, we measure the diffusion of software vulnerability using the link prediction between software vulnerabilities. To do this, we implement the Node2Vec algorithm. Node2Vec is a machine learning algorithm that computes the vector coordinates of a node in a network. In brief, Node2Vec implements an objective function which maximizes the log-probability of two different nodes are linked together. In computational form, Node2Vec computes $\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$, where $N_S(u)$ is the random sampling strategy S of neighborhood network N. Based on the Node2Vec, we calculate the feature matrix of $G(V, E)$. For instance, two nodes u and v with the corresponding feature vectors $f(u)$ and $f(v)$ that generate $g(u, v)$ such that $g: V \times V \rightarrow R^{d'}$. The d' is the size of the feature representation (i.e., dimension) for the pair of (u, v). The link $g(u, v)$ can be represented using the Hadamard product (i.e., $g(u, v) = f_i(u) \cdot f_i(v)$, for each i^{th} node in G) (Grover and Leskovec 2016; Li et al. 2017). Therefore, if the machine learning model succeeds in learning the pattern of $f_i(u) \cdot f_i(v)$, the machine learning model can predict the link between u and v by calculating prediction score. For the machine learning model, we used logit, which performed better than supportive vector machine (SVM) and feed forward neural net (FFNN)[2]. Based on the machine learning, we count diffusive nodes by codifying the nodes as 0 for non-linked nodes and 1 for linked nodes in G. Finally, we calculate *Diffusion* using the proportion of the number of diffusive software vulnerabilities divided by the total number of software vulnerabilities in network G.

[2] We used receiver operating characteristics to compare the performance using the baseline of Adamic Adar (2003). We report ROC: logit(0.95), SVM(0.80), FFNN (0.66), and Adamic Adar (0.55).

Third, we calculate network centrality measures based on the mathematical forms. A software vulnerability may have a larger influence over the neighborhood software vulnerability because it has more external digital resources that are shared by other entities in the digital ecosystem. Using mathematical notation, we define *degree centrality* as an explanatory variable which is the return value of the function $\deg(v)$ in the network of $G(V,E)$. The *degree centrality* refers to the counted number of edges incident upon a node. The function counts the edges incident to the node v. We divide the *degree centrality* by statistics freedom (i.e., total number of nodes – 1) to normalize the number of nodes. We calculate: $degree\ centrality_v = \frac{\deg(v)}{C}$, where C is the number of v in $G(V,E)$.

However, *degree centrality* is limited in its ability to capture the severity of a software vulnerability caused by global network relationships of software vulnerabilities. For instance, a group of software vulnerabilities could be linked to other groups through mediating nodes. A mediating node could have more pervasive influence compared with other nodes. This may because the mediating node could trigger one software vulnerability to other popular software vulnerabilities (e.g., those that have more edges). By doing so, the software vulnerability could be discovered and triggered by the mediating nodes. In this case, degree centrality cannot properly estimate the impact of the global linking. The method called *eigenvector centrality* provides ways to calculate this type of centrality. We calculate the *eigenvector centrality* as: $eigenvector\ centrality_v = \lambda \sum_M^C A_{MV} \cdot \frac{\deg(x)}{C}, v \neq x$. $A_{G(V,E)}$ is the adjacent matrix, where A_{ux} is 0 when the edge (v,x) does not exist (i.e., negative edge) and 1 when the edge does exist. λ is the eigenspace coefficient that is the value of linear transform L of eigenvector (v,x). In network notation, we define the λ as $V_\lambda = \{v \in V : L_v = \lambda_v\}$.

The degree and eigenvector centralities regard local and global edge relationships. However, those measures do not consider the density of the edge structure that can be an alternative measure for network centrality. In social network research, the clustering coefficient is used as a measure of the tendency of each node to cluster together. The *clustering coefficient* indicates how individuals are embedded in their neighborhood. Software vulnerabilities could be associated and interact with each other by sharing overlapping APIs which contain vulnerable codes. Some shared vulnerable codes do not create a malfunction for one software program, but the shared code could initiate or cause another software program to malfunction. This means that if the software vulnerability lies in a highly dense edge structure, it could have more chances to be exposed to other vulnerabilities, leading to more severe software vulnerability. To measure this tendency, we follow the guidance of Opsahl and Panzarasa (2009) and reflect the weighted edges for the clustering coefficient. They suggest a computational equation to consider the effects of dense edges in a global network as:

$$clustering\ coefficient_v = \frac{\Sigma_{number\ of\ closed\ triplets_v} w}{\Sigma_{number\ of\ all\ triplets_v} w}$$

EMPIRICAL APPROACH AND DISCUSSION

We used a linear model estimation approach to estimate the effects of the independent variables on the dependent variable. The unit of analysis is software vulnerability. We specify the following equation for the ordinary least square (OLS) regression model [3] :

$$\Psi = B_0 + B_1 diffusion + B_2 \ln degree\ centrality + B_3 eigenvector\ centrality + B_4 clustering\ coefficient + Controls + \Omega.$$

Ψ is our dependent variable (i.e., *severity of software vulnerability*), and Ω is the error terms of OLS. The controls include a time dummy. We use the log transform for degree due to over-

[3] We used a capital Greek letter to avoid confusion with other equations.

dispersion and to allow for a more intuitive interpretation of the regression coefficient. The unit of the number of solutions and the number of affected products is 10. The unit for the reported time duration is 10 days. We checked the correlation and variance inflation factors and found no major issues (corr. < 0.2 and VIF < 10.0). We included a summary of the descriptive statistics in the appendix[4]. We report the OLS results in Table 1.

Table 1. Empirical Results

DV. Software Vulnerability Score	(1)	(2)
Model	**Control**	**Main**
Diffusion		3.363*** (0.187)
Ln(Degree Centrality)		0.179*** (0.00658)
Eigenvector Centrality		-0.783*** (0.0356)
Clustering Coefficient		-0.639*** (0.0430)
#Solution	0.00489* (0.00261)	0.00401 (0.00245)
Reported time duration	0.0195*** (0.000423)	0.0111*** (0.000629)
#Affected Product	0.0532*** (0.0156)	0.0338** (0.0132)
Constant	6.536*** (0.0466)	3.312*** (0.170)
Time Dummy	Yes	Yes
Observations	85,520	85,520
***R*-squared**	0.202	0.315

********p* **< 0.01; *****p* **< 0.05; ****p* **< 0.10; robust standard errors are reported in parenthesis.**

In control (model 1), we present a baseline knowledge contribution model in which we only include the control variables. We found that all the control variables were significant except #affected product. Since the control variables include a time consideration (i.e., reported time duration), the positive inclination effects may be interpreted as cumulative effects on the severity of software vulnerability.

[4] VIF and Correlation tables are also available upon request

We found support for P1 because the diffusion is consistently positive and highly significant in our main model (model 2). In the main model, the coefficient estimate suggests that the output coefficient of diffusion is 3.363 ($p < 0.01$), which means that an increase of one unit of diffusion possibility leads to an increase of 3.363 in software vulnerability severity. The results indicate that a software vulnerability that has more chances to be diffused) would be more likely to be more severe. We also found that P2 was supported by examining the significances of degree centrality (0.179, $p < 0.01$), eigenvector centrality (-0.783, $p < 0.01$) and clustering coefficient (-0.639, $p < 0.01$).

Degree centrality is positively associated with software vulnerability. This means that locally and densely related software vulnerabilities would be more severe. However, when network centrality is considered in light of global network effect, the results were opposite to those for direct centrality. First, eigenvector centrality measures the extent of connectivity to popular software vulnerabilities (e.g., a node has many degrees) in the digital ecosystem. The negative association could be interpreted to mean that software vulnerabilities that are linked to a popular software vulnerability were already fixed by the solutions to the popular software vulnerability. This could result in less severity of software vulnerabilities. Second, the clustering coefficient measures the tendency of each software vulnerability to cluster together. The result showed a negative impact on the severity of software vulnerability. This result may indicate that the developers related software vulnerability in the digital ecosystem collectively act to reduce the possible errors and vulnerable codes in the shared APIs. As a result, the connected software vulnerabilities in a group could have less severity.

CONCLUSION

The object of this study is to understand the impact of the diffusion of software vulnerability and network centrality in a digital ecosystem on the severity of software vulnerability. The diffusion of software vulnerability is positively associated with the severity of software vulnerability. In addition, we confirmed that different measures of network centralities influence the severity of software vulnerability in different ways. Our research found empirical evidence of the complex relationships between network centrality and efforts of the community to reach solutions. The results lead to diverse interpretations that the position of software vulnerability in a digital ecosystem could have different effects on other co-located and global neighbor software vulnerabilities. These characteristics should be studied in more detail in future research.

REFERENCES

Alhazmi, O. H., Malaiya, Y. K., and Ray, I. 2007. "Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems," *Computers & Security* (26:3), pp. 219-228.

Applewhite, A. 2004. "Whose Bug Is It Anyway? The Battle over Handling Software Flaws," *IEEE Software* (21:2), pp. 94-97.

Barros, A. P., and Dumas, M. 2006. "The Rise of Web Service Ecosystems," *IT Professional* (8:5), pp. 31-37.

Brada, P., and Jezek, K. 2015. "Repository and Meta-Data Design for Efficient Component Consistency Verification," *Science of Computer Programming* (97), pp. 349-365.

Breukers, Y. P. 2016. "The Vulnerability Ecosystem: Exploring Vulnerability Discovery and the Resulting Cyberattacks through Agent-Based Modelling,").

Decan, A., Mens, T., and Grosjean, P. 2019. "An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems," *Empirical Software Engineering* (24:1), pp. 381-416.

Gopalakrishna, R., Spafford, E., and Vitek, J. 2005. "Vulnerability Likelihood: A Probabilistic Approach to Software Assurance," *CERIAS, Purdue Univeristy Technical Reports* (6), p. 2005.

Grover, A., and Leskovec, J. 2016. "Node2vec: Scalable Feature Learning for Networks," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*: ACM, pp. 855-864.

Kula, R. G., German, D. M., Ouni, A., Ishio, T., and Inoue, K. 2018. "Do Developers Update Their Library Dependencies?," *Empirical Software Engineering* (23:1), pp. 384-417.

Lagerström, R., Baldwin, C., MacCormack, A., Sturtevant, D., and Doolan, L. 2017. "Exploring the Relationship between Architecture Coupling and Software Vulnerabilities," *International Symposium on Engineering Secure Software and Systems*: Springer, pp. 53-69.

Li, L., Wang, W., Yu, S., Wan, L., Xu, Z., and Kong, X. 2017. "A Modified Node2vec Method for Disappearing Link Prediction," *2017 IEEE 15th International Conference on Dependable, Autonomic and Secure Computing, 15th International Conference on Pervasive Intelligence and Computing, 3rd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*: IEEE, pp. 1232-1235.

Luo, J., Lo, K., and Qu, H. 2014. "A Software Vulnerability Rating Approach Based on the Vulnerability Database," *Journal of Applied Mathematics* (2014), 932397.

Medeiros, I., Neves, N., and Correia, M. 2015. "Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining," *IEEE Transactions on Reliability* (65:1), pp. 54-69.

Moore, S., Armstrong, P., McDonald, T., and Yampolskiy, M. 2016. "Vulnerability Analysis of Desktop 3D Printer Software," *2016 Resilience Week (RWS)*: IEEE, pp. 46-51.

Nguyen, V. H., and Tran, L. M. S. 2010. "Predicting Vulnerable Software Components with Dependency Graphs," *Proceedings of the 6th International Workshop on Security Measurements and Metrics*: ACM, p. 3.

Opsahl, T., and Panzarasa, P. 2009. "Clustering in Weighted Networks," *Social Networks* (31:2), pp. 155-163.

Phonemom. 2017. "Cost of Cyber Crime Study," *Accenture*), p. 1:56.

Rahimi, S., and Zargham, M. 2013. "Vulnerability Scrying Method for Software Vulnerability Discovery Prediction without a Vulnerability Database," *IEEE Transactions on Reliability* (62:2), pp. 395-407.

Rehman, S., and Mustafa, K. 2009. "Research on Software Design Level Security Vulnerabilities," *ACM SIGSOFT Software Engineering Notes* (34:6), pp. 1-5.

Rubin, B., and Pollet, A. 2017. "2016 FINRA Analysis: A Record-Breaking Year for Fines," *Journal of Investment Compliance* (18:2), pp. 1-8.

Shahriar, H., and Zulkernine, M. 2012. "Mitigating Program Security Vulnerabilities: Approaches and Challenges," *ACM Computing Surveys (CSUR)* (44:3), 11.

Sosa, M. E., Mihm, J., and Browning, T. R. 2013. "Linking Cyclicality and Product Quality," *Manufacturing & Service Operations Management* (15:3), pp. 473-491.

Tevis, J.-E. J., and Hamilton, J. A. 2004. "Methods for the Prevention, Detection and Removal of Software Security Vulnerabilities," *Proceedings of the 42nd Annual Southeast Regional Conference*: ACM, pp. 197-202.

Um, S., and Yoo, Y. 2016. "The Co-Evolution of Digital Ecosystems,").

Walden, J., Stuckman, J., and Scandariato, R. 2014. "Predicting Vulnerable Components: Software Metrics Vs Text Mining," *2014 IEEE 25th International Symposium on Software Reliability Engineering*: IEEE, pp. 23-33.

Wang, T., Wei, T., Gu, G., and Zou, W. 2010. "Taintscope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection," *2010 IEEE Symposium on Security and Privacy*: IEEE, pp. 497-512.

Watson, R. N., Woodruff, J., Neumann, P. G., Moore, S. W., Anderson, J., Chisnall, D., Dave, N., Davis, B., Gudka, K., and Laurie, B. 2015. "Cheri: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization," *2015 IEEE Symposium on Security and Privacy*: IEEE, pp. 20-37.

Yoo, Y., Lyytinen, K. J., Boland, R. J., and Berente, N. 2010. "The Next Wave of Digital Innovation: Opportunities and Challenges: A Report on the Research Workshop 'Digital Challenges in Innovation Research'," *Available at SSRN 1622170*.