# Cast to Vote: A Socio-technical Network Analysis of an Election Smartphone Application

JULIA STACHOFSKY, Department of Management, Information Systems, and Entrepreneurship, Washington State University
ASSEFAW GEBREMEDHIN, School of Electrical Engineering and Computer Science, Washington State University
ROBERT E. CROSSLER, Department of Management, Information Systems, and Entrepreneurship, Washington State University



This article builds a socio-technical dataset and associated network and provides exploratory analysis of a GitHub repository for a smartphone application used by the Elizabeth Warren campaign during the 2020 Iowa Democratic Caucuses. It provides insights into the types of technology used by campaigns for tracking primary delegates and the social actors that have potential influence over those technologies. Results suggest that certain nodes in the network have more influence over the election technology and that security risks can manifest through the unique interaction between technical and social network components as supply chain attacks. The work sheds light on implications of how election technology is audited and designed, especially for potential public-facing voting technology. A metric for evaluating socio-technical vulnerability is also proposed.

CCS Concepts: • **Applied computing** → **Voting/election technologies**; • **Social and professional topics** → **Socio-technical systems**; • **Security and privacy** → **Social aspects of security and privacy**;

Additional Key Words and Phrases: e-Vote, voting, democracy, election security, social network analysis

**ACM Reference format:**
Julia Stachofsky, Assefaw Gebremedhin, and Robert E. Crossler. 2022. Cast to Vote: A Socio-technical Network Analysis of an Election Smartphone Application. *Digit. Gov.: Res. Pract.* 3, 1, Article 3 (March 2022), 17 pages.
https://doi.org/10.1145/3501031

## 1 INTRODUCTION

In February of 2020, the Iowa Democratic Party provided an unintentional case study of how technology failure can impact an election. The Iowa Democratic Caucuses results were delayed, in large part due to a smartphone

A. Gebremedhin is supported in part by NSF Award No. IIS-1553528.
Authors' addresses: J. Stachofsky and R. E. Crossler, Washington State University Pullman, Todd Hall 442, PO Box 644743, WA 99164-4743; emails: julia.stachofsky@wsu.edu, rob.crossler@wsu.edu; A. Gebremedhin, School of Electrical Engineering and Computer Science, PO Box 642752, Washington State University, Pullman, WA 99164-2752; email: assefaw.gebremedhin@wsu.edu.

2639-0175/2022/03-ART3 $15.00
https://doi.org/10.1145/3501031

application used to record delegates producing inconsistent results [13]. Beyond the technical failure, the smartphone application development company, Shadow Inc., did little to assuage rampant conspiracy theories.

The case of the delegate tracking smartphone application used by the Iowa Democratic party differs from other voting systems, because election officials are using the technology to record delegate counts rather than voters casting votes. In the caucus system, voters are counted based on which candidate's precinct captain they stand by. Delegates are then calculated based on the number of voters at each precinct captain's station. However, there is still incentive for attackers to influence delegate counts as the momentum gained from winning early caucuses and primaries can boost the performance of a candidate through the remaining primary elections [18, 32]. Furthermore, a modification of the delegate counts changes the will of the voters.

The official smartphone application used has not been open-sourced, but a similar application built by the Elizabeth Warren campaign for delegate tracking was recently made open-source on GitHub [37]. To re-iterate, the application that failed in the Caucus is NOT the same as the Warren application, but it presents a unique opportunity to study the socio-technical network structure of an election technology. As such our research is guided by the two following research questions.

1. *How can open-source applications be modeled as socio-technical networks?*
2. *What insights related to securing election technologies can be gleaned from a socio-technical perspective that a purely social or a purely technical perspective lack?*

The first question is related to general methodologies in the developer social networks literature. The extant literature on developer social networks has primarily modeled networks as either purely technical or purely social [20, 33]. We propose a method by which multi-modal socio-technical modeling [16] can be used to integrate the two distinct technical and social networks into a single network where interactions between technical and social actors can be modeled and analyzed. The second question is more applied in nature: it deals with using the developed network model within the digital government space. The results of this analysis will be useful as a method for evaluating the threat landscape of election technologies, particularly around potential supply chain attacks, as well as provide the groundwork for potential analyses that can be used as part of the election auditing processes.

The next section discusses the background of socio-technical network analysis and the events that transpired at the Iowa Democratic Caucuses in 2020. This is followed by a description of our research method and a presentation of the socio-technical network we constructed. The article closes with results, discussion of the findings, and concluding remarks.

## 2 BACKGROUND

### 2.1 Socio-technical Systems Theory

The theoretical base for the development of the analyzed network is socio-technical systems theory as established by Bostrom and Heinen [3, 4]. Their work primarily focuses on understanding the interaction between technical and social subsystems in organizations and the role it plays in the development of information systems. It emerges as a theory to understand why information systems fail or are otherwise problematic, which at the time was often attributed to only social *or* technical causes [3]. It should be noted that this perspective treats the two subsystems as a distinct relational ontology, as opposed to the socio-material theoretical lens, which would suggest that the social and technical cannot be meaningfully separated [24].

Within socio-technical systems theory the technical system consists of the technology artifacts and the tasks that are being conducted with the artifacts. Within the social subsystem there are individuals and the structures in which they operate. The subsystems interact with each other, but there are also interactions within each subsystem (such as between technology and task) [3]. It is through these inter- and intra-system interactions that emergent behaviors and outcomes are best understood.

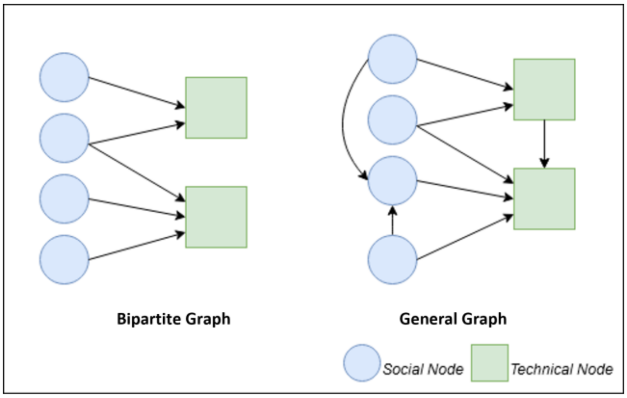


Fig. 1. Bipartite graph example.

Socio-technical systems theory is the foundational lens of the information systems field [29]. The core need to understand both the social and technical aspects of a system has, however, also been applied in computer science [11, 19] and more recently digital government [8, 12, 15, 34]. To fully understand a socio-technical system could require deep qualitative work, however a significant leap can still be made by adapting these principles to more abstract quantitative analysis. Our article adapts these general socio-technical concepts to add richness to typical single modal networks analysis and integrates them with network analysis in the vein of Kane and Alavi's [16] work on socio-technical network interactions.

## 2.2 Socio-technical Network Analysis

Methodologically this work builds upon the GitHub developer social networks literature in computer science [20, 33]. In the GitHub social networks literature, repositories are understood to construct social connections in the network rather than explicitly separate artifacts that developers interact with [20]. Technical-to-technical connections have been studied separately as well in Thung et al. [33], where both developer-to-developer networks and project-to-project networks are studied.

We expand this analysis into the information systems literature by modeling the network as a socio-technical network rather than a purely social or purely technical network. Kane and Alavi [16] use a socio-technical multimodal network to study user-system interactions and their implications for firm performance. Our article differs in the treatment of the technical aspect. Kane and Alavi [16] treat the technical artifact as a single node embedded in a social network, whereas our approach breaks the technical artifact into a network of technical nodes that have social networks attached to them.

From this theoretical base the network that is developed is a graph that consists of *social* nodes and *technical* nodes. However, this is not modeled as a bipartite graph. Bipartite graphs require that each edge (or link) connects nodes of different types (in this case technical and social), meaning that technical nodes cannot connect to technical nodes and similarly social nodes cannot connect to social nodes [9]. Instead, we use the multimodal perspective as used by Kane and Alavi [16].

Figure 1 shows an example of a bipartite graph (left) versus a general directed graph (right) involving social and technical nodes. Since the graph at the right has connections between members of the same node type, it is not a bipartite graph. We chose a general graph rather than a bipartite graph so that we can model connections between technical nodes, social nodes, and socio-technical connections. By taking a multi-modal approach rather than a bipartite graph approach, we can model interactions between different technical nodes in the network. For example, the relationship between a library dependency and the parent application that calls a library. Within the

social subsystem, we can also model interactions between different social nodes. For example, the relationship between a developer and their followers.

### 2.3 The 2020 Iowa Democratic Caucuses

*2.3.1 How the Caucus System Works.* Caucuses operate differently than a typical election. Rather than a tallying of votes for a preferred candidate, the focus of a primary is on acquiring state delegate equivalents. Unlike typical elections where voter anonymity is important [22], voters are counted based on which candidate's precinct captain they stand by at the precinct location.

The caucuses proceed in two rounds. After the first round, if a candidate has 15% or less of the total people at a precinct location standing in their area, then that candidate will not be allocated any delegates [21]. In the next round voters have the option to re-sort to different candidates for the final count for delegate allocation [21]. While voters could technically all retain their original choices, many voters will shift to a candidate that was above the 15% threshold in the first round so that they can allocate more delegates to their second-choice candidate. In addition to the caucus smartphone application, a paper backup of the voter counts is maintained for each caucus site [21]. Iowa is the first primary caucus each year, although only 41 total delegates of the 1,991 total delegates across the rest of the United States needed to win the primary are available for allocation [21].

*2.3.2 What Happened in 2020?* For the 2020 election, the Iowa Democratic Party used a smartphone application for reporting precinct results in the Iowa caucus. Concerns were raised a month before the election took place as the Iowa Democratic Party withheld technical details about the smartphone application, resulting in less scrutiny from outside cybersecurity professionals [14]. Additional concerns were raised about Internet connectivity issues, technology literacy of precinct leaders, as well as just the overall insecure ecosystem of running a web-connected smartphone application downloaded onto personal android devices of precinct leaders for logging delegate counts [26].

The night of the caucuses proved to be a disaster. This was not due to a cybersecurity attack like many feared, but rather a failure of the technology itself [36]. Delegate count reports from precincts were inconsistent and data was missing, calling into question all information reported from the smartphone application [1]. This led to the party reverting to paper records of voter tallies, resulting in a three-day delay before finally announcing Pete Buttigieg as the winner of the most delegates [1]. This smartphone application failure potentially had significant implications for the primary elections, as momentum gained from wining early caucuses and primaries can boost the performance of a candidate through the rest of the primary elections [18, 32].

Post-election security audits indicate that the Iowa Democratic party were lucky to walk away with only a technology failure. Officials at Veracode conducted a technical audit of the smartphone application afterwards and found a major vulnerability; data being transmitted from the phone over the Internet lacked basic safeguards and could be modified in transit [14]. There is no evidence of this occurring in practice, but it would have been a major security risk had additional state caucuses adopted the smartphone application.

## 3 RESEARCH METHOD

### 3.1 The Elizabeth Warren Campaign Delegate Smartphone Application

Figure 2 provides an overview of the Warren campaign's smartphone application. The figure shows only one precinct but would be the same setup for different precincts across Iowa. The lower box represents a precinct location with a precinct captain that enters delegate counts from the delegate worksheet into the smartphone application. The upper box on the left represents the Warren team's web infrastructure, and the box on the right represents a Google sheet hosted on Google's web infrastructure used for storing and querying precinct location data. Places where arrows flow over a cloud indicate that information is being transmitted over the Internet.

The caucus smartphone application is a **Progressive Web Application (PWA)** [35]. PWAs are not run in a web-browser tab but are instead added to a user's mobile device home screen in an app-like fashion. PWAs

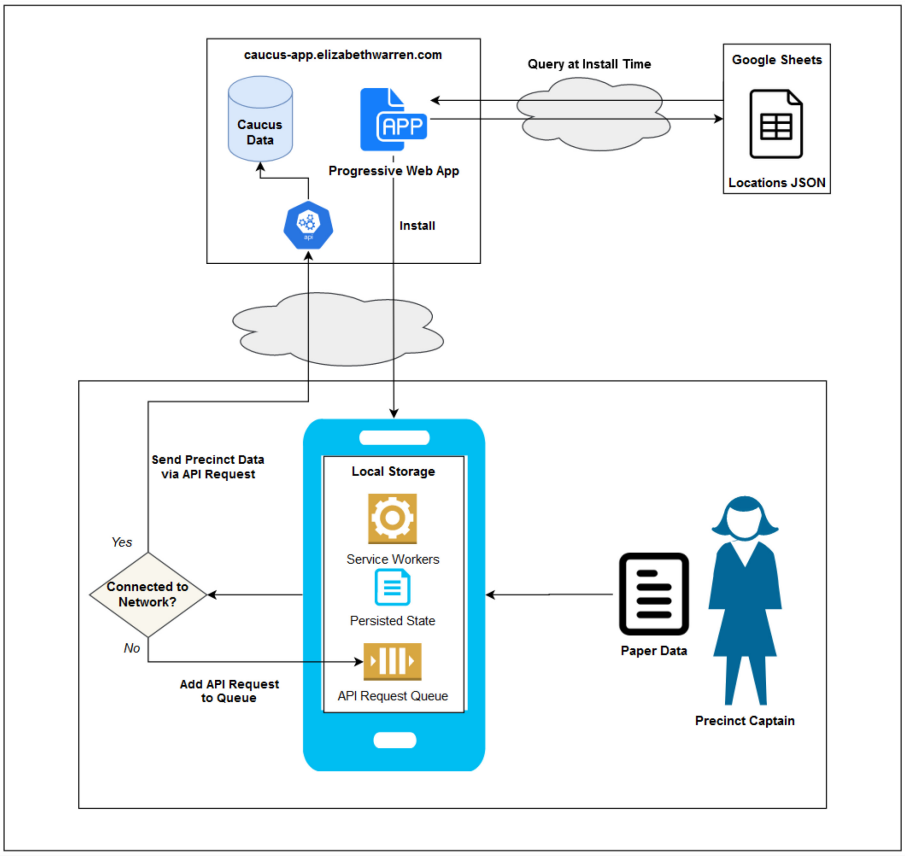


Fig. 2. Warren caucus smartphone application overview.

provide functionalities typically only available from a natively installed application through the web application and they lack a lot of the capabilities of regular mobile/desktop applications [2, 28]. When the application is built, a query to a Google sheet with precinct location data is made as the locations were changing often leading up to the caucuses [37]. When precinct captains navigate to the web address of the application it is installed on their mobile device. This is done so if the device loses connectivity, then API requests with new delegate data can be queued in local storage until an Internet connection is reestablished [37].

## 3.2 Dataset Construction

*3.2.1 Technical Nodes.* A novel dataset was constructed for this research. The source of the dataset was the Iowa Caucus App JavaScript repository published on GitHub by the Elizabeth Warren campaign team [37]. We used the package.json file to model the technical dependencies of the smartphone application as visualized by the dependency graph tool NPMGraph [17] (Figure 3). This tool visualizes the packages that the repository references, as well as the dependencies of those packages. Production code release dependencies only were used in this study.

Based on this visualization, we created a CSV file consisting of connections between the 21 dependencies and the caucus smartphone application, resulting in a total of 22 unique technical nodes in the network. Direction of arrows were reversed from the NPMGraph output, however, as we are modeling influence directionality. In particular, directions in our modeling show that it is the dependencies that influence the caucus smartphone

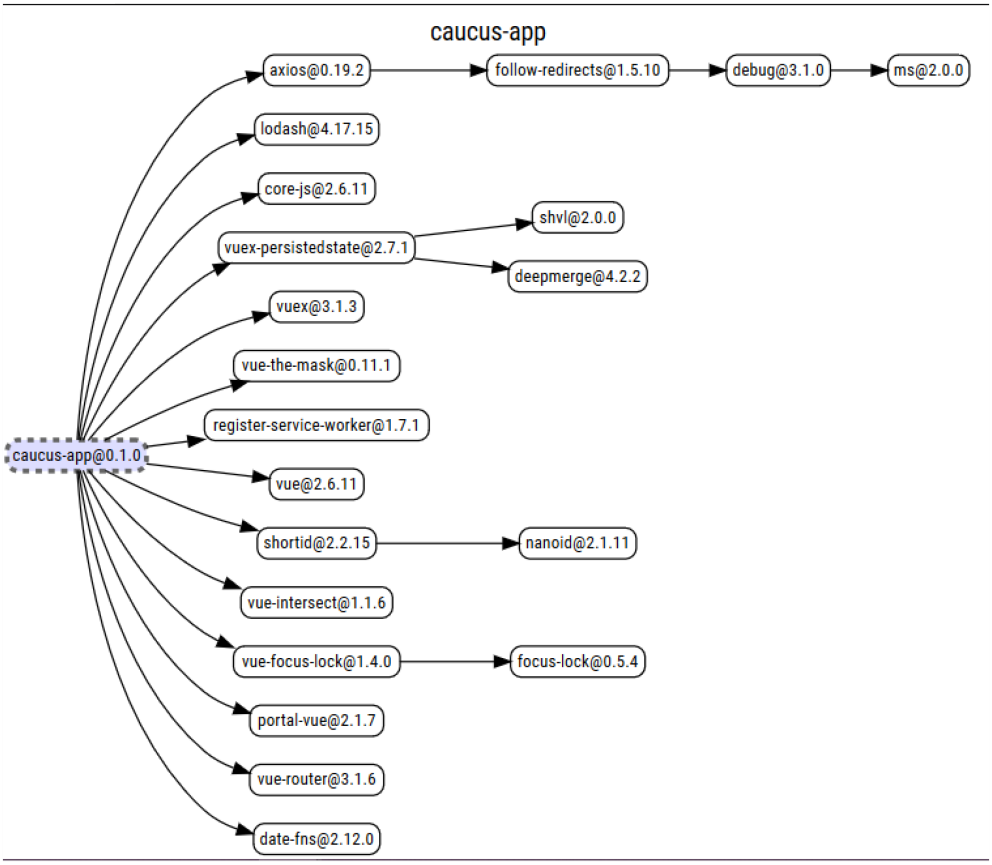


Fig. 3. Package dependencies for Elizabeth Warren delegate smartphone application (Generated with NPMGraph [17]).

application, not the caucus smartphone application that influences the dependencies. Last, we analyzed the package.json file with the tool NPM Audit [38] to find known vulnerabilities in the dependencies that are used in the smartphone application.

*3.2.2 Social Nodes.* Initially, we sought to collect the contributor to repository connections using the GitHub API, however it was not possible to extract contributor lists without push access to a given repository. Therefore, we instead used the Python library Beautiful Soup [39] to scrape HTML from the following URL where **repo_owner** and **repo_name** are substituted with the repository that contributors are being extracted from: https://github.com/repo_owner/repo_name/issues/show_menu_content?partial=issues/filters/authors_content.

This URL returns an html page with every user who has ever contributed to the repository. From there, we extracted the usernames with Beautiful Soup. Unfortunately, this web scraping method provides less data about the social nodes. For example, there is no weight to the connections based on how many contributions any given user has added to the repository. While some users may have only one contribution to the repository, others may have many. All social nodes were anonymized as "@S#" where # is a number arbitrarily assigned to a social node.

For the social-to-social connections, we used the GitHub API to pull the follower list and following list of each contributor extracted with Beautiful Soup. We restricted the social network to who the contributors were following, and the followers of the contributors. The node sets of followers, following, and contributors are

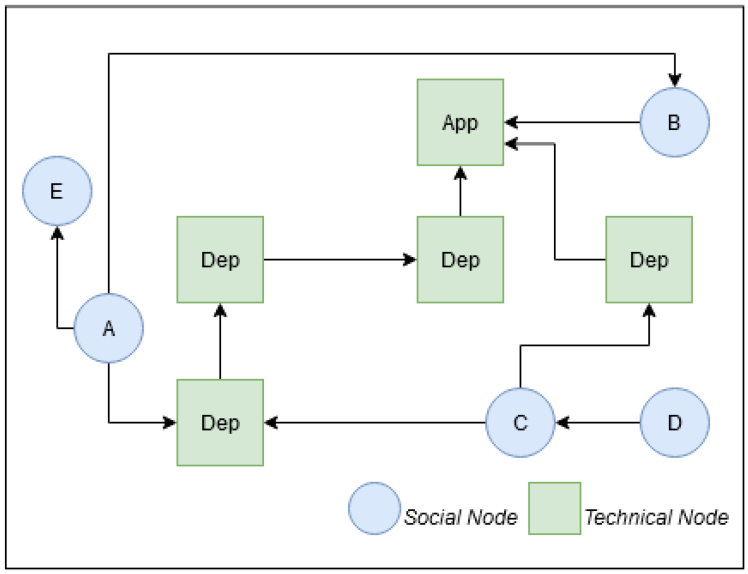


Fig. 4. Socio-technical network example.

not mutually exclusive (for example, a contributor to one repository might follow a contributor on another repository). Instead, these are all different contextual relations within the overall set of social nodes represented by directed edges between the nodes. For example, a directed edge from a social node, node 1, to another social node, node 2, would be a following relationship from the perspective of node 1 and a follower relationship from the perspective of node 2.

*3.2.3 Network Construction.* The CSV files containing the social nodes, technical nodes, and connections were parsed with Python and constructed into a directed graph object using the NetworkX library [40]. The graph object was then exported as a GML file for future analysis. This GML file along with all the analysis code has been made available online [31]. In addition to NetworkX and Beautiful Soup, the third-party libraries igraph [41], NumPy [42], Matplotlib [43], and Pandas [44] were used to conduct data cleaning and data analysis.

Figure 4 shows an example of how a socio-technical application network could look. The App node represents the application repository. The Dep nodes represents the repositories for all the dependencies the application relies on. The chain of dependencies could only be one repository, or it could be hundreds if not thousands. App and Dep are technical node types. The nodes A, B, and C are developers who have contributed to the repositories. The nodes D and E represent developers who have not contributed directly to any repositories but are following or being followed by developers that contribute to the repositories. The nodes A, B, C, D, and E are social node types.

A developer can have direct influence on the application repository (*B*) if they are one of the developers of the application. Developers can also have indirect influence mediated through other technical nodes (*A* and *C*) of varying lengths. Developers can also contribute to more than one repository (*C*). The direction of the arrows indicates the influence the developer and the technical node has on the next level up. It is also possible that developers could gain access to *App* through social networks as well (relationship between A and B).

## 3.3 Metrics and Notations

Number of nodes (vertices): Recall that nodes in our graph model consist of two subsets, technical and social nodes:

$$n,\ n_{social},\ n_{technical}.$$

Table 1. Descriptive Statistics of the Initial Network

| $n$ | $n_{technical}$ | $n_{\text{social}}$ | $m$ | $Deg_{\text{min}}$ | $Deg_{\text{max}}$ | $Deg_{\text{avg}}$ | $l_{\text{avg}}$ | $d$ |
|---|---|---|---|---|---|---|---|---|
| 267 | 22 | 245 | 666 | 1 | 121 | 4.989 | 0.224 | 0.009 |

Table 2. Centrality Statistics of the Initial Network

| Centrality Measure | Node | Score |
|---|---|---|
| $C_{\text{d}}$ | date-fns_date-fns | 0.455 |
| $C_{\text{in}}$ | date-fns_date-fns | 0.451 |
| $C_{\text{out}}$ | @S211 | 0.113 |

Number of edges (connections): Note that edges are directed:

$$m.$$

Average Shortest Path Length: Average path length between any pair of nodes in the graph. Let $V$ be the set of nodes in the graph $G$, $d(s,t)$ be the shortest path from node $s$ to node $t$, and $n$ be the number of nodes in $G$:

$$l_{avg} = \frac{\sum_{s,t \in V} d(s,t)}{n(n-1)}.$$

Density: Ratio of number of edges in the graph with respect to the maximum possible number of edges. Let $n$ be the number of nodes and $m$ be the number of edges in graph $G$:

$$d = \frac{m}{n(n-1)}$$

Degree Centralities: Aggregate count of incoming, outgoing and total (the sum of these two) edges connected to a node normalized by the maximum degree in the network. These metrics are denoted as follows:

$$C_d, C_{in}, C_{out}.$$

The next section will discuss the results of the analysis.

## 4 RESULTS

The results are presented in three sections. First, we provide descriptive statistics and a visualization of a network consisting of repositories and immediate contributors. The second section presents the results of the technical vulnerability analysis conducted using the node package manager *audit function*. The final part of our analysis is a socio-technical analysis of the supply chain attack risks of the application.

### 4.1 Network Descriptive Statistics

Table 1 summarizes the descriptive statistics for the network. The range of values for the network density $d$ is 0 to 1, with 0 representing a graph with no edges, and 1 representing a complete graph. The initial network consisted of 22 technical nodes, and 245 repository contributors connected by 666 edges. The average path length between nodes was low at 0.224 and the density was at 0.009.

Table 2 summarizes the degree centrality statistics for the network. The centrality measures include the most central node as well as the calculated centrality score for that node. Social nodes are denoted with an @ symbol. Two of the centrality measures, degree centrality and in-degree centrality, identified the technical node date-fns_date-fns as the most central to the network, and the out-degree centrality measure identified the user @S211, a social node, as the most central node to the network with a combination of being a contributor to three repositories and following 27 developers in the network.

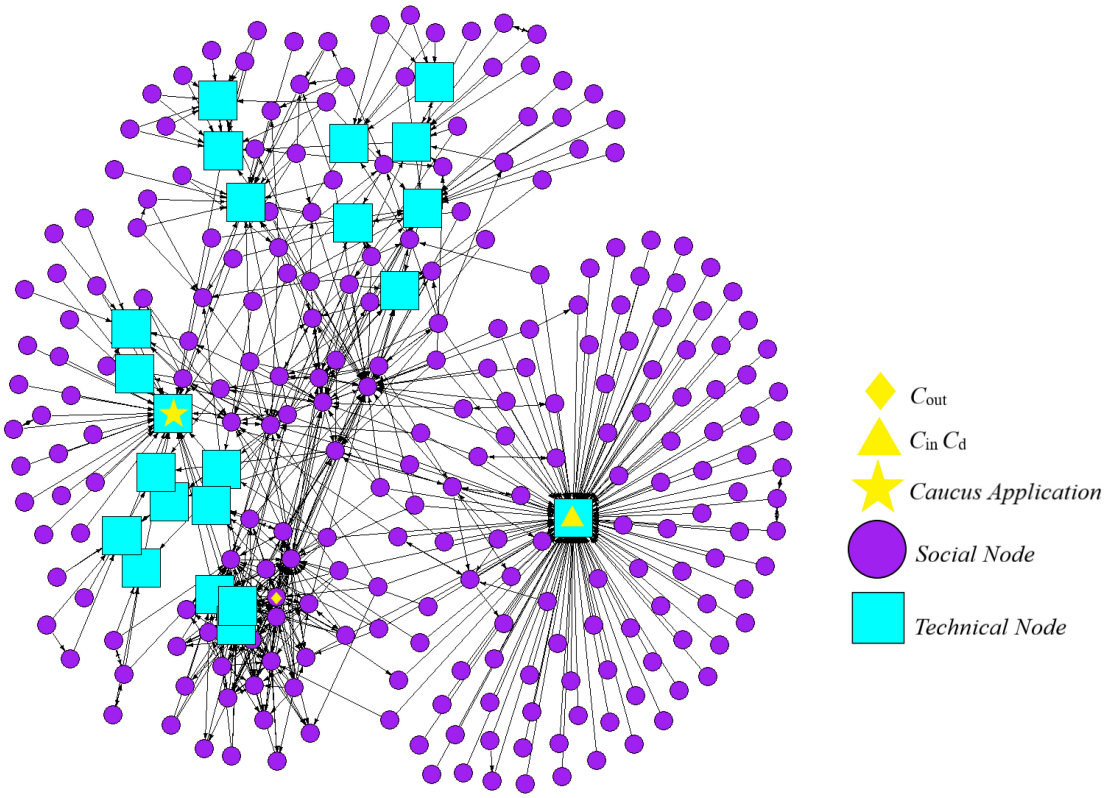


Fig. 5. Iowa caucus smartphone application initial network.

Table 3. Node Clustering of the Initial Network

| **Cluster** | 4 | 11 | 2 | 1 | 7 | 9 | 10 | 3 | 6 | 12 | 5 | 13 | 14 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Node Count** | 105 | 36 | 31 | 17 | 17 | 11 | 11 | 10 | 10 | 6 | 4 | 4 | 3 | 2 |

Figure 5 shows a diagram of the entire network generated from the dataset. Purple circles represent social nodes and cyan squares represent technical nodes. In this network, we see the contributors of each repository, social connections between the contributors, and the interconnections between dependencies that make up the smartphone application. Arrows represent directionality of relationship. If an arrow points from one social node to another, then that means that developer follows another developer. If an arrow points from one technical node to another, then that means that the technical repository is a dependency of another technical repository. If an arrow points from a social node to a technical node, then that means the developer contributes to that repository.

In addition to plotting the graph, we have conducted a community detection (clustering) analysis to identify subgraphs of densely connected nodes. The clustering analysis algorithm used was the cluster_walktrap [27] function in igraph [41]. The cluster_walktrap function finds communities through random walks, with the assumption that short random walks stay in the same community. From the 267 nodes, we find 14 different clusters presented in Table 3. The cluster with the most nodes had 105 nodes, and the cluster with the least number of nodes had 2 nodes. Node clustering is relevant in this context as more connected groups may adhere to social norms more than individual actors less worried about losing social capital [5].

### 4.2 Technical Vulnerability Analysis—Node Package Manager Audit

Production dependencies are the dependencies actually needed in order for the smartphone application to run on the device [45]. For this application there were 21 production dependencies, which we conducted a node package manager audit on. These dependencies are the same as the repositories modeled in our socio-technical network. Of the 21 dependencies, we found three vulnerabilities affecting two of the dependencies. Two of the vulnerabilities were of high threat severity, and one was of low threat severity.

The low severity vulnerability is security advisory for a prototype pollution bug in lodash, a JavaScript utility library [46]. An attacker can cause a denial of service or execute code remotely by modifying the prototype of Object in the function zipObjectDeep [46]. We searched the code repository of the caucus smartphone application and found no instances where zipObjectDeep was used meaning this vulnerability would not have been exploitable during the election. One of the high severity vulnerabilities is a security advisory for a command injection bug also in lodash [47]. This vulnerability opens the application to the risk of executing arbitrary commands and can be initiated remotely without physical device access. The exploit potential is unknown as this vulnerability is not tied to an instance of a specific function. Both the low and high severity vulnerabilities for lodash were published many months after the Iowa caucus on July 1, 2020, and May 6, 2021, respectively.

The last high severity security vulnerability is a security advisory for a server-side request forgery bug for axios, a JavaScript HTTP client library [48]. This vulnerability allows for an attacker to bypass any proxies configured for the application. When searching through the code repository, we did not see any usage of a proxy server for the caucus application, suggesting that this vulnerability would not have been exploitable during the election. Like the two lodash vulnerabilities this vulnerability was not discovered until after the election with the advisory published on January 4, 2021.

### 4.3 Socio-technical Vulnerability Analysis—Supply Chain Attack

The previous section looked at examples of software components becoming vulnerable through design flaws and code errors. While still useful, this analysis is limited in the insight it offers for vulnerabilities of indirect nature. In this section, we address that limitation by shifting the focus to malicious attacks intentionally brought upon a software component. One such example of a malicious attack relevant to the open-source software community is *supply chain attacks*. Supply chain attacks are when attackers target components within the overall supply chain with the intent to impact the upstream applications that use those components [23]. Such an attack can have a widespread impact given that many organizations rely on open-source software packages to build their applications. To model this attack for the smartphone application, we start with identifying technical nodes with the most influence on other technical nodes in the network, filtering the network to only include technical-to-technical connections. The results of this analysis are presented in Figure 6.

Unsurprisingly, the Iowa caucus application is the most central node in the technical network, given that most of the repositories listed here are direct library imports to build the application. Each of these imports is a potential for a supply chain attack with direct influence on the smartphone application. Another repository with a high level of influence is the vuex-persisted state module. It has the second highest in degree centrality, meaning that there are multiple modules within the network it relies on. The node also has outgoing connections to other repositories making it a high-risk node.

For a supply chain attack to be initiated a social actor must maliciously engage said attack. As such, this purely technical analysis does not entirely capture the supply chain attack risk of the application. Thus, we conduct additional analyses that include the social nodes in the network. We first look at technical nodes with the highest number of incoming connections from social nodes, filtering the network to only include social-to-technical connections. We also include nodes with high technical dependency with few social actors, inspired by an event that happened in 2016. A developer unpublished all their node modules from the node package manager over

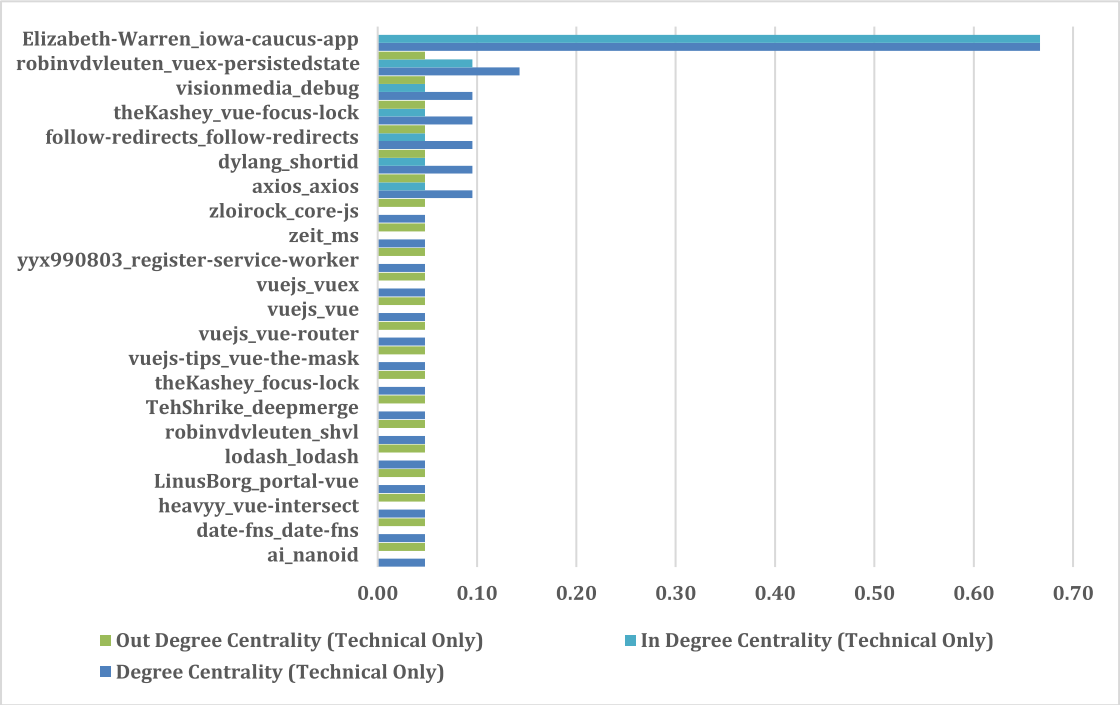


Fig. 6. Technical nodes only centrality.

disagreements related to one of their module names having the same name as a company threatening to sue over trademark infringement. One of the modules they unpublished was *left-pad*, a simple module of only 11 lines of code, but that module was a build dependency for the React.js ecosystem, causing outages and errors across the world [6]. Unlike a supply chain attack, this action was not malicious but rather an unintended consequence of the many interdependencies on third-party software applications. Returning to our case, the results of our analysis along a similar line of thought are presented in Figures 7 and 8.

Which nodes have the most influence in the network changes when looking at the socio-technical connections only. Three of the top five nodes were ranked the lowest in the technical-only analysis, including the most prominent node the *date-fns* module. These nodes with the highest in-degree centrality show that they have the highest number of developers contributing to the code repository. This increases the potential number of actors that can inject malicious code into the repository. However, at the same time these repositories could be under additional scrutiny due to the fact that there are more developers working on the project. The cluster (Table 3) to which a social actor belongs may indicate the risk profile associated with a given technical node or social actor, with larger clusters more likely to adhere to social structure and norms [5]. One node that is in the top five across both analyses is the *axios* module, meaning that it has higher influence on other technical nodes while also having proportionally more developers contributing to its codebase.

The previous analysis identifies technical nodes with the most social actors influencing their development and at-risk technical nodes with the least number of social actors. Also of interest are social actors that have influence over multiple technical nodes in the network. For this analysis, we filter the network to only the social-to-technical connections. The results of this analysis are presented in Figure 9, where only nodes connected to three or more repositories are presented as there are 245 nodes in total.

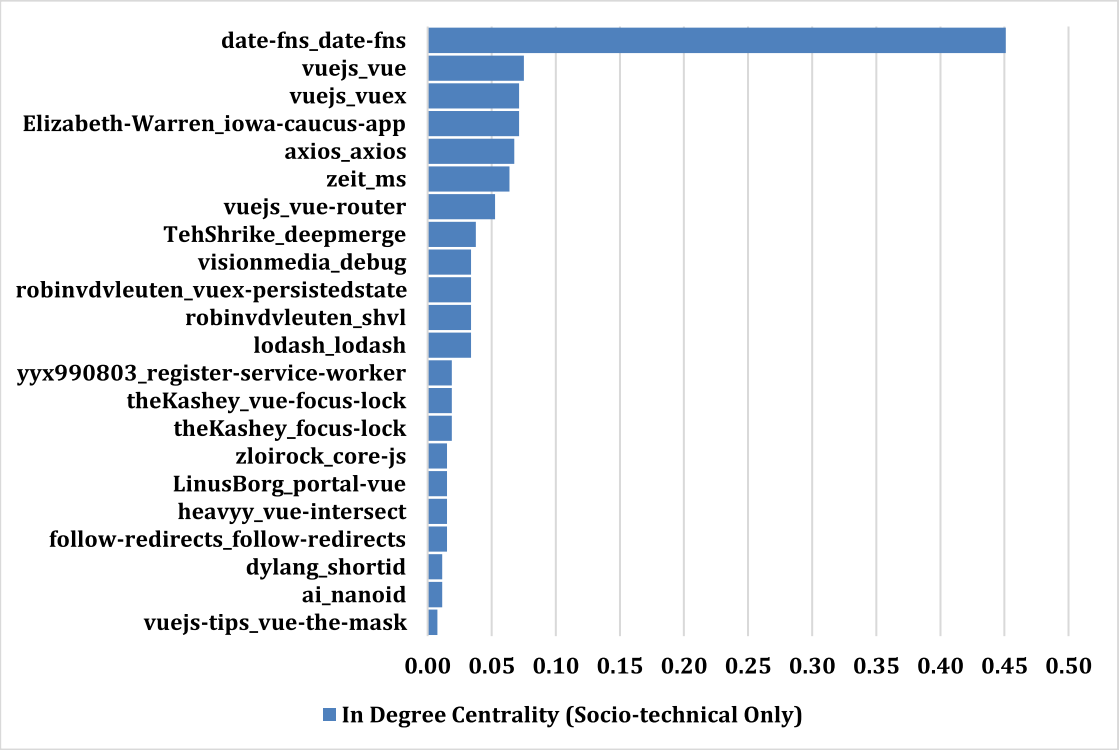


Fig. 7. Socio-technical connections only centrality.

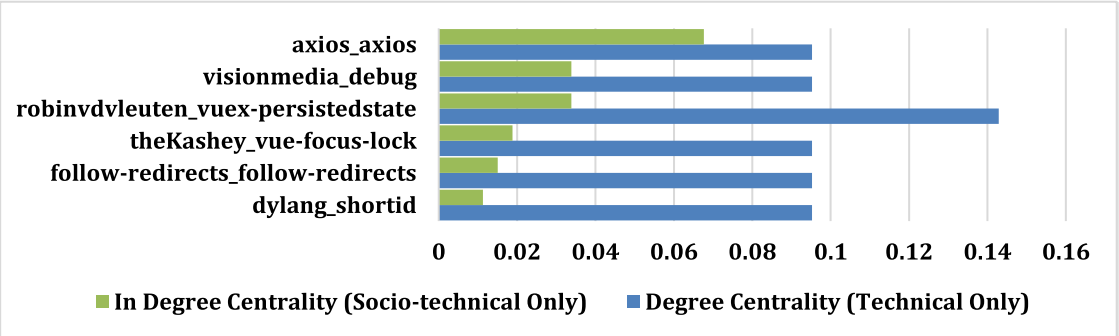


Fig. 8. High technical influence low social actors.

The results of this analysis show that some of the social nodes within the network are contributing to multiple repositories. This is not completely unexpected, given that many of the packages included in the application are related to the *Vue.js* JavaScript framework. The social node with the most influence had contributions to 16 of the repositories in the network, with the next highest coming in at 6 and 4 repository contributions, respectively. From a supply chain attack perspective, this means that the social actors have more venues to inject potentially malicious code.

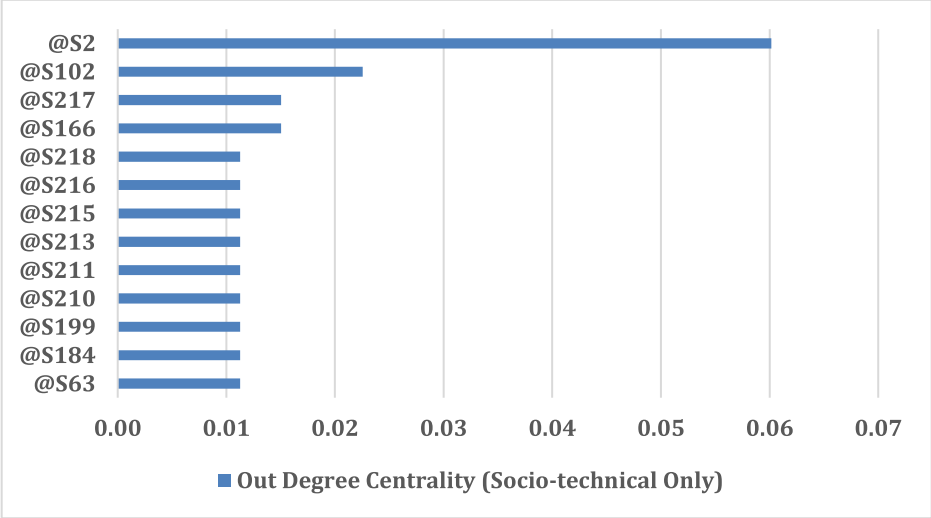


Fig. 9. Social nodes with the highest technical influence.

### 4.4 Proposed Network Metric for Identifying Socio-technical Vulnerability

Based on the socio-technical analysis above, we propose a metric that can be used to identify the socio-technical vulnerability score for a network. We present the formula as follows. Let $V$ be the set of nodes in the graph $G$, $S$ be the set of social nodes, $T$ be the set of technical nodes, and $n$ be the number of nodes in $G$. Let $Cst_{in}(u)$ be the social-to-technical in-degree centrality for a technical node u. Let $Ctt_d(v)$ be the technical-to-technical degree centrality for a technical node v.

$$STV = \frac{\sum_{u,v \in T} Cst_{in}(u) + Ctt_d(v)}{n(n-1)}.$$

The metric includes a summation of these centrality measures for each technical node in the network and is averaged to produce an overall socio-technical vulnerability score for the network. When adding technical components, the new STV can be compared to the previous STV to determine the socio-technical risk posed by adoption of that component.

The social-to-technical in-degree centrality measure captures the risk brought in from new social actors in the network. With the adoption of a new technical component, there are now more developers that contribute to that node, but they may also contribute to other nodes, as we established in Figure 9. The technical-to-technical degree centrality measure captures the risk brought in from technical inter-dependencies. The component may be isolated, and thus its failure or exploitation does not have broader impacts on the network. However, a technical component may be connected to many other technical components in the network increasing the risk.

## 5 DISCUSSION

### 5.1 Network Analysis

In the overall network, we see a more prominent role of technology as having central influence in the network. The repository *date-fns_date-fns* was identified as the most central technical node, which we see in the network structure as it has the most developer nodes connected to it. This repository is a direct dependency for the smartphone application, meaning developers that contribute to *date-fns_date-fns* are not too far removed to potentially influence the smartphone application. However, it should be noted that with additional developers

contributing to the repository, a more rigorous review process conducted by different developers may be in place that would make it harder to influence the smartphone application negatively. One social node, @S211 was identified as central based on out-degree centrality measures. In the context of the initial network, this suggests that this user has the highest combination of repositories they contribute to, and users that they follow that are contributing to the technical dependencies in the network. This may suggest that this user could exert influence through their social connections in the network in addition to the technical dependencies that they contribute code to.

The node package manager audit provides insight on the dynamic nature of security threats. Each of the vulnerabilities identified for the election application were not identified until after the election had concluded. This highlights the importance of rigorous post-election audits to identify anomalies, as at the time of technology deployment these vulnerabilities may not be known by the security community. Additionally, even once these vulnerabilities are identified actually resolving them can take a long time to be patched and deployed into production [7].

The supply chain vulnerability analysis is where the value of a socio-technical analysis is realized most in this research. Since supply chain attacks are inherently socio-technical, requiring a malicious social actor to exploit vulnerable code [23], both the social and technical subsystems must be modeled to evaluate risk. Our analysis of social nodes that are connected to multiple technical nodes highlights the malicious influence potential through multiple vectors. A purely social network analysis would not identify the relation to the attack vector, and a purely technical analysis does not account for additional influence given to an existing actor in the dependency network.

We also identified technical nodes that had many social actors contributing to them. With the additional developers comes the potential risk of malicious code being injected. A purely social perspective would be able to identify actors in the network, but not the attack vectors they have influence over. A purely technical analysis could identify existing vulnerabilities in an adopted software package but gives no insight into how many developers are working on the project. When working on something as security critical as election technology, being aware of the number of social actors in the overall dependency network is an important risk to evaluate. We also identified technical nodes with high influence that had few social actors contributing to them. While there are fewer potential malicious actors in this scenario, we highlight the outsized influence individuals connected to these technical components have.

## 5.2 Implications for Researchers

For network researchers, one contribution is providing an example of modeling multi-modal socio-technical networks from GitHub data. While a multi-modal network may not be necessary for every context, our analysis shows the additional analyses and insights that can be derived by including nodes of different types with relations across social and technical boundaries. We also build upon the work of Kane and Alavi [16] by decomposing a singular technical node into a combination of technical nodes. This allows for more complex socio-technical networks to be modeled and could be continually decomposed to the level that is necessary for a given study. Our research stopped at the level of package modules, but other researchers may consider modeling technical artifacts at the function level for example. We also propose a metric for evaluating socio-technical centrality. Future work should empirically validate this metric beyond the scope of this single application network.

For digital government research, we contribute to literature on election IT artifacts. This research gives insight into both the technical structure of an election technology and also the social subsystem it interacts with. Understanding the structure of a system helps in identifying potential weak points that can be exploited by malicious actors, which is of heightened valence in contexts such as elections. Future research can build upon this by searching for generalizations across different IT artifacts used in digital government. Combining deep theorizing

of the political and social context as well as theorizing of the IT artifact itself is critical for understanding the interactions of these systems moving forward.

### 5.3 Implications for Practice

For practice, this research emphasizes the caution that must be applied when developing election technologies. In a sector like election technology where security is paramount it is critical that additional scrutiny beyond functionality of a library is considered. Organizations should also consider the contributors to a given repository and the broader supply chain risks when integrating components into their applications. It is unrealistic to expect all election technologies to be written from scratch without outside resources, and even then, that is no guarantee of a secure system. However, the code developed for this research project may serve as a starting point to further develop auditing and risk modeling tools to help organizations make informed decisions about third-party software components. We have made all our code free and available on GitHub for any researchers or organizations that wish to utilize this analysis.

We generally agree with the National Academies' consensus that electronic voting cannot be conducted safely in the United States with current technologies [22]. As such, we urge organizations and governments to consider whether they should be developing technologies like the application analyzed here in the first place. If, however, these technologies become more commonplace within United States elections [10, 25], then at the very least robust paper backups and post-election audits should be conducted. Taking a socio-technical approach to problems of election security will be critical for securing election infrastructure as more technologies are implemented in the future.

### 5.4 Limitations and Future Work

One limitation of our work is that we primarily focus on the smartphone application components. Additional threat vectors related to data in transit, devices that the smartphone application is executed on, and election officials at the caucuses would also need to be considered to get a full understanding of the election security context. It should be noted that our model is only *part* of the complex socio-technical network, not the entirety of the network. Future research could include additional interactions in the socio-technical network beyond developers and software components. There is also an assumption that adding external dependencies increases security and failure risks. While increased complexity of a software artifact often negatively affects software reliability [30], the risk could be offset by the actual contents of the software package. Future research could refine this approach by evaluating the type of software component included.

Last, we only included binary connections as part of our analysis. Future work could include additional characteristics of the social nodes and technical nodes of the network. For example, a developer that has contributed more to a repository has more opportunities to influence the network than a developer that has only contributed once. Similarly, an application may use a technical dependency but only in a limited capacity where its influence in the network is less than other dependencies.

## 6 CONCLUSION

This research serves as a first step toward modeling the socio-technical nature of election technology. Through the development of a novel dataset, applied to a multimodal network structure, we have gained insight into the actors that can influence election technology and the risks, social and technical, that emerge when adding components to a software artifact. This is an important context, as those who influence election technology have the potential to influence elections themselves. We also contribute to the literature on open-source network modeling and supply chain attacks by proposing a method of integrating both social and technical actors. Election technology developers must give significant thought to whether to adopt a dependency into their application, as not only are the security vulnerabilities of that technology now embedded in the election technology but so are the developers who maintain those dependencies.

## ACKNOWLEDGMENTS

We thank the editor and the anonymous reviewers for their comments. This article benefitted greatly from their feedback on previous versions. We also thank Helen Catanese for her help with the network visualization code on this project and James Halvorsen for his feedback on an earlier version of this article.

## REFERENCES

[1] Amanda Becker and Michael Martina. 2020. Buttigieg narrowly wins Iowa caucuses: State party results. Retrieved from https://www.reuters.com/article/us-usa-elections-idUSKBN2002JS.

[2] Andreas Biørn-Hansen, Tim A. Majchrzak, and Tor-Morten Grønli. 2017. Progressive Web Apps: The possible web-native unifier for mobile development. In *Proceedings of the 13th International Conference on Web Information Systems and Technologies, SCITEPRESS Science and Technology Publications*. 344–351. DOI: https://doi.org/10.5220/0006353703440351

[3] Robert P. Bostrom and J. Stephen Heinen. 1977. MIS problems and failures: A socio-technical perspective. Part I: The Causes. *MIS Quarterly* 1, 3 (Sep. 1977), 17. DOI: https://doi.org/10.2307/248710

[4] Robert P. Bostrom and J. Stephen Heinen. 1977. MIS problems and failures: A socio-technical perspective, Part II: The application of socio-technical theory. *MIS Quarterly* 1, 4 (Dec. 1977), 11. DOI: https://doi.org/10.2307/249019

[5] James Coleman. 1988. Social capital in the creation of human capital. *Amer. J. Sociol.* 94, (1988), 27.

[6] Keith Collins. 2016. How one programmer broke the internet by deleting a tiny piece of code. *Quartz*. Retrieved from https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/.

[7] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 181–191. DOI: https://doi.org/10.1145/3196398.3196401

[8] José van Dijck and Bart Jacobs. 2020. Electronic identity services as sociotechnical and political-economic constructs. *New Media Soc.* 22, 5 (May 2020), 896–914. DOI: https://doi.org/10.1177/1461444819872537

[9] David Easley and Jon Kleinberg. 2010. *Networks, Crowds, and Markets*. Cambridge University Press, Cambridge.

[10] Zach England. 2020. West Virginia officials want other states to adopt online voting for deployed troops. *Military Times*. Retrieved from https://www.militarytimes.com/news/2020/07/21/west-virginia-officials-want-other-states-to-adopt-online-voting-for-deployed-troops/.

[11] Katayoun Farrahi, Remi Emonet, and Alois Ferscha. 2012. Socio-technical network analysis from wearable interactions. In *Proceedings of the 16th International Symposium on Wearable Computers*. IEEE, 9–16. DOI: https://doi.org/10.1109/ISWC.2012.19

[12] Tian-peng Gao, Hong Su, and Ting Yu. 2021. The connotation and logical construction of government digital transformation—based on the analysis of sociotechnical system theory. In *Proceedings of the E3S Web Conference. 03069*. DOI: https://doi.org/10.1051/e3sconf/202125103069

[13] Ginger Gibson. 2020. Explainer: Election meltdown: What went wrong at the Iowa caucuses. Retrieved from https://www.reuters.com/article/us-usa-election-iowa-problems-explainer-idUSKBN1ZY2B9.

[14] Jack Gillum and Jessica Huseman. 2020. The Iowa caucuses app had another problem: It could have been hacked. Retrieved from https://www.propublica.org/article/the-iowa-caucuses-app-had-another-problem-it-could-have-been-hacked?token=gKXJz8oaquJjtQQXfWohwVdXFUbpvysX.

[15] David Jamieson, Rob Wilson, and Mike Martin. 2020. Is the GaaP wider than we think?: Applying a sociotechnical lens to Government-as-a-Platform. In *Proceedings of the 13th International Conference on Theory and Practice of Electronic Governance*. ACM, 514–517. DOI: https://doi.org/10.1145/3428502.3428580

[16] Gerald C. Kane and Maryam Alavi. 2008. Casting the Net: A multimodal network perspective on user-system interactions. *Info. Syst. Res.* 19, 3 (Sep. 2008), 253–272. DOI: https://doi.org/10.1287/isre.1070.0158

[17] Robert Kieffer. 2017. *NPMGraph*. Retrieved from http://npm.broofa.com/.

[18] Brian Knight and Nathan Schiff. 2010. Momentum and social learning in presidential primaries. *J. Political Econ.* 118, 6 (2010), 41.

[19] Hana Kopackova and Petra Libalova. 2017. Smart city concept as socio-technical system. In *Proceedings of the International Conference on Information and Digital Technologies*. 8.

[20] William Leibzon. 2016. Social network of software development at GitHub. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'16)*. IEEE, 1374–1376. DOI: https://doi.org/10.1109/ASONAM.2016.7752419

[21] Domenico Montanaro. 2020. How the Iowa caucuses work—and why they're important. Retrieved from https://www.npr.org/2020/01/30/800588703/how-the-iowa-caucuses-work-and-why-theyre-important.

[22] National Academies of Sciences, Engineering, and Medicine. 2018. *Securing the Vote: Protecting American Democracy*. National Academies Press, Washington, D.C. DOI: https://doi.org/10.17226/25120

[23] Marc Ohm, Arnold Sykosch, and Michael Meier. 2020. Towards detection of software supply chain attacks by forensic artifacts. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. ACM, 1–6. DOI: https://doi.org/10.1145/3407023.3409183

[24] W. J. Orlikowski. 2010. The sociomateriality of organisational life: considering technology in management research. *Cambridge J. Econ.* 34, 1 (Jan. 2010), 125–141. DOI: https://doi.org/10.1093/cje/bep058
[25] Miles Parks. 2019. In 2020, Some Americans will vote on their phones. Is that the future? Retrieved from https://www.npr.org/2019/11/07/776403310/in-2020-some-americans-will-vote-on-their-phones-is-that-the-future.
[26] Miles Parks. 2020. Despite election security fears, Iowa caucuses will use new smartphone app. Retrieved from https://www.npr.org/2020/01/14/795906732/despite-election-security-fears-iowa-caucuses-will-use-new-smartphone-app.
[27] Pascal Pons and Matthieu Latapy. 2005. Computing communities in large networks using random walks (long version). Retrieved from http://arxiv.org/abs/physics/0512106.
[28] Sam Richard and Pete LePage. What are Progressive Web Apps? Retrieved from https://web.dev/what-are-pwas/.
[29] Suprateek Sarker, Sutirtha Chatterjee, Xiao Xiao, Copenhagen Business School, Amany Elbanna, and Royal Holloway University of London. 2019. The sociotechnical axis of cohesion for the IS discipline: Its historical legacy and its continued relevance. *MISQ* 43, 3 (Jan. 2019), 695–719. DOI: https://doi.org/10.25300/MISQ/2019/13747
[30] Norm Schneidewind and Mike Hinchey. 2009. A complexity reliability model. In *Proceedings of the 20th International Symposium on Software Reliability Engineering*. IEEE, 1–10. DOI: https://doi.org/10.1109/ISSRE.2009.10
[31] Julia Stachofsky. 2021. dgov-sociotechnical-election-network. Retrieved from https://github.com/pacpix/dgov-sociotechnical-election-network.
[32] Wayne P. Steger. 2008. Forecasting the presidential primary vote: Viability, ideology, and momentum. *Int. J. Forecast.* 24, 2 (Apr. 2008), 193–208. DOI: https://doi.org/10.1016/j.ijforecast.2008.02.006
[33] Ferdian Thung, Tegawendé F. Bissyandé, David Lo, and Lingxiao Jiang. 2013. Network structure of social coding in GitHub. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*. 323–326. DOI: https://doi.org/10.1109/CSMR.2013.41
[34] Heike Vornhagen, Brian Davis, and Manel Zarrouk. 2018. Sensemaking of complex sociotechnical systems: the case of governance dashboards. In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*. ACM, 1–2. DOI: https://doi.org/10.1145/3209281.3209392
[35] Warren for President Tech Team. 2020. Open-source Tools from the Warren for President Tech Team. *Medium*. Retrieved from https: //medium.com/@teamwarren/open-source-tools-from-the-warren-for-president-tech-team-f1f27d2c7551.
[36] Zack Whittaker. 2020. A result reporting app used the Iowa caucus has crashed. Retrieved from https://social.techcrunch.com/2020/02/03/iowa-caucus-app-failed/.
[37] Github. 2020. Elizabeth-Warren/Iowa-caucus-app. Retrieved from https://github.com/Elizabeth-Warren/iowa-caucus-app.
[38] npm-audit. 2018. Retrieved from https://docs.npmjs.com/cli/audit.html.
[39] Beautiful Soup. 2004. Retrieved from https://www.crummy.com/software/BeautifulSoup/.
[40] NetworkX. 2014. Retrieved from https://networkx.github.io/.
[41] igraph—Network analysis software. 2006. Retrieved from https://igraph.org/.
[42] NumPy. 1995. Retrieved from https://numpy.org/.
[43] Matplotlib: Python Plotting. 2003. Retrieved from https://matplotlib.org/.
[44] Pandas—Python Data Analysis Library. Retrieved from https://pandas.pydata.org/.
[45] Yarn. Types of dependencies. 2008. Retrieved from https://yarnpkg.com/en/docs/dependency-types/.
[46] Prototype Pollution—Lodash. npmjs Advisories. 2020. Retrieved from https://www.npmjs.com/advisories/1523.
[47] Command Injection—Lodash. npmjs Advisories. 2021. Retrieved from https://www.npmjs.com/advisories/1673.
[48] Server-Side Request Forgery—Axios. 2021. Retrieved from https://www.npmjs.com/advisories/1594.

Received October 2020; revised August 2021; accepted November 2021