Towards Socio-Technical Topology-Aware Adaptive Threat Detection in Software Supply Chains

Thomas Welsh, Kristófer Finnsson, Brynjólfur Stefánsson and Helmut Neukirchen
Department of Computer Science, University of Iceland, Reykjavík, Iceland
{tomwelsh, kdf2, brs, helmut}@hi.is

***Abstract*—Software supply chains (SSCs) are complex systems composed of dynamic, heterogeneous technical and social components which collectively achieve the production and maintenance of software artefacts. Attacks on SSCs are increasing, yet pervasive vulnerability analysis is challenging due to their complexity. Therefore, threat detection must be targeted, to account for the large and dynamic structure, and adaptive, to account for its change and diversity. While current work focuses on technical approaches for monitoring supply chain dependencies and establishing component controls, approaches which inform threat detection through understanding the socio-technical dynamics are lacking. We outline a position and research vision to develop and investigate the use of socio-technical models to support adaptive threat detection of SSCs. We motivate this approach through an analysis of the XZ Utils attack whereby malicious actors undermined the maintainers' trust via the project's GitHub and mailing lists. We highlight that monitoring technical and social data can identify trends which indicate suspicious behaviour to then inform targeted and intensive vulnerability assessment. We identify challenges and research directions to achieve this vision considering techniques for developer and software analysis, decentralised adaptation and the need for a test bed for software supply chain security research.**

***Index Terms*—software supply chain, socio-technical, adaptation, topology, threat detection**

I. Introduction

The software supply chain (SSC) involves a complex network of interdependent components from initial requirements gathering, to deployment upon an end-user system. Modern software development heavily employs code reuse to enhance productivity, this results in code making use of a network of dependencies, most of which are opaque to the developer and user. This interdependency ensures attacks against one component may propagate downstream [1] with SSC attacks, such as malicious packages, increasing by 156% yearly [2]. These components are under continuous evolution and independently maintained by different organisations and individuals operating within various geographical locations and legislative environments [1]. This requires threat detection and security controls to evolve alongside them and to perform continuous evaluation within varying social and legislative environments. SSC attacks have been highlighted as a key challenge by the security and software engineering communities [3] due to the increasing number of high-profile incidents [4], [5], their impact upon a wide number of end-users, and the challenges in detecting attacks relating to the complexity of the SSC.

Typically, approaches to secure the SSC focus upon dependency monitoring, enhancing secure development practices, or applying controls to each supply chain stage. These approaches do not aim to mitigate social attack vectors and their effectiveness in mitigating them is unknown [3], [6].

In contrast, this paper proposes an alternative approach which leverages the complexity characteristics of the SSC, as opposed to building controls to mitigate them. Through monitoring and adapting to changes in: social behaviours, components and structural properties, we suggest that threat detection can operate in a targeted manner to detect complex and evolving social threats. We propose to generate and monitor *socio-technical topologies of the SSC* to support analysis of socio-technical threat indicators. To manage change, these topologies can be analysed within an adaptive framework, resulting in *Socio-Technical Adaptive SSC Threat Detection*. Expanding on a previous approach for cyber-physical supply chains [7], yet applying techniques for modelling and analysing socio-technical dynamics specific to software development.

Towards achieving this vision, the contributions of this position paper are as follows: 1) We motivate the need for socio-technical topology modelling and analysis to support threat detection of SSCs through a study of the XZ Utils attack [8]. We highlight that aggregating social and technical data sources could indicate suspicious developer behaviour. 2) We propose a framework to adaptively detect threats in SSCs supported by socio-technical topologies.

The rest of this paper is structured as follows. Section II provides a background in SSC security, socio-technical security, and socio-technical modelling. A motivating example of the XZ Utils SSC attack is presented in Section III. Section IV presents the proposed approach, *Socio-Technical Adaptive SSC Threat Detection*. Section V provides a discussion, followed by a conclusion in Section VI.

II. Background and Related Work

Dependency-focused attacks vectors are common to SSC security [3]. The Software Bill of Materials (SBOM) is a key mitigator but suffers from numerous challenges [3], [9]. Version control systems, build systems, package distribution systems, and end-user machines are also targets of attack [10]. Threats from social factors are also numerous, e.g. malicious insiders injecting code [11] or from a variety of psychological phenomena [12]–[14]. Consequently, software repositories can be mined for social indicators of code quality, [15], [16], bug reports can predict vulnerabilities [17], [18], version control systems and mailing lists can link developers with

code [19] and chats can indicate project sentiment [20]. Where anomalous code deviations, sentiment or bug reports indicate a vulnerable software component. Machine learning (ML) approaches show success in monitoring individual components but not the complete supply chain or its change. They may also suffer from high rates of false positives [21]. Quantum ML has shown to offer lower accuracy and higher computation than classical approaches [22]. The wide social and technical attack surface suggests that SSCs should be considered as a socio-technical system [23], [24]. Threats arising from vulnerabilities in human, social and organisational factors are widely known, e.g. social engineering [25], inadequate organisational incident response preparation [26], and ineffective security culture [27]. Socio-technical security considers that unique vulnerabilities arise due to the interplay of both social and technical vulnerabilities [28]. To identify unique socio-technical threats, software-based threat detection must have the means to model and reason about these concepts cohesively.

Socio-technical systems are complex and therefore challenging to model, monitor, and analyse due to their emergent, heterogeneous, dynamic and large structures [29]. Network-theoretic approaches are frequently used to understand these structures [30]. Topology modelling is used to combine multiple dimensions, (e.g. cyber, physical, social) to allow cross-dimensional analysis of structural properties [7], [31], [32]. However, there is a current lack of techniques which investigate the use of socio-technical topologies of SSCs.

Pervasive monitoring and inspection of all components across a dynamically evolving SSC is cost ineffective, creating a resource allocation problem [7]. While topologies can be used to manage the structural complexity of a complex system, further techniques are needed to handle this change and emergent behaviours. Adaptive software is used to manage change, e.g. based on the Monitor, Analyse, Plan, Execute, Knowledge (MAPE-K) adaptive software framework [33], and was previously used for threat detection in cyber-physical supply chains [7]. We propose this approach can also apply to threat detection in SSCs yet including novel techniques for modelling socio-technical factors of developers to inform targeted analysis of software components for vulnerabilities.

III. Motivating Example: The XZ Utils Supply Chain Attack And Threat Indicators

In this section, we present an initial analysis of the social and technical factors related to the XZ Utils attack [34], [35].

A. XZ Utils Attack Stages

The XZ Utils attack can be categorised into several stages: **Legitimate Contributions (LC) Late '21 – Early '22:** Author *Jia Tan* begins contributing small patches through the xz-devel mailing list, focusing on minor bug fixes and feature improvements. In total, *Jia Tan* authored over 500 patches in multiple GitHub projects. These early contributions appear legitimate, gradually establishing credibility.
Escalation of Control (EC) Early '22 – Early '24: Lead maintainer *Lasse Collin* is placed under sustained social pressure via mailing list discussions. Multiple anonymous or sockpuppet accounts criticize slow project progress, gradually pressuring him into relinquishing control. **Mid '22 – Early '23:** *Jia Tan* is recognized in Git metadata as an author, eventually becoming a co-maintainer with commit access. **March '23:** *Jia Tan* tags and builds the first release under own control (*v5.4.2*), marking a critical shift in project leadership. **June '23 – January '24:** Technical groundwork is laid for the attack: **June '23:** *Hans Jansen*, a new actor, submits patches introducing the GNU indirect function (*ifunc*) feature, which is later leveraged for the backdoor. **July '23:** *Jia Tan* disables ifunc support in OSS-Fuzz builds, reducing external visibility. **January '24:** Control over the XZ Utils website is transferred to GitHub Pages, under *Jia Tan's* control.
Backdoor Deployment (BD) February '24 – March '24: *Jia Tan* introduces a subtly obfuscated backdoor into the codebase, hidden within binary test input files. The README file discourages detailed scrutiny, making detection difficult.
Exposure and Removal (ER) March 28, '24: Security researcher *Andres Freund* identifies anomalous SSH behaviour and privately notifies *Debian* and *RedHat*. **March 29, '24:** *RedHat* assigns *CVE-2024-3094* [36], marking the vulnerability as a critical security risk. **March 30, '24:** A public backdoor warning is issued, and *Fedora* confirms that compromised XZ Utils versions were included in a recent release.

B. Socio-Technical Threat Indicators

The complex technical backdoor was achieved through social engineering by *Jia Tan* who was seemingly acting as a normal developer [stage LC, see subsection III-A]. This poses the question *What types of developer behaviour can be used to indicate the software is trending towards an insecure state?* To study this, we mined the XZ Utils GitHub commits and issues [37] and mailing list [38] from the start of the project in 2008 to 2025. We examined various sources of developer activity, focusing on the threat actor, *Jia Tan*, with a comparison with known legitimate maintainers. Examining commit metrics, Table I presents statistics for file-level Git commit statistics between *Jia Tan* and *Lasse Collin* for the period 2022-01 to 2024-06. Despite *Lasse Collin* affecting more files, *Jia Tan* made, on average, more substantial changes to each file than *Lasse Collin*.

Figure 1 plots the average overall changes over time and clusters of the attack stages are highlighted: the most substantial changes occurred when backdoor was inserted [BD][ER]. The results illustrate that at similar times to the anomalous

TABLE I
File-Level Line Change Statistics: Lines added/deleted by commits from Jia Tan vs. Lasse Collin from 2022-01 to 2024-06

Statistic	Jia Tan	Lasse Collin
Total File Changes	697	1973
Average Additions	89.42	28.26
Average Deletions	42.10	18.31
Average Total Changes	131.53	46.56
Std Dev Additions	396.20	146.41
Std Dev Deletions	163.45	147.61
Std Dev Total Changes	492.14	249.01

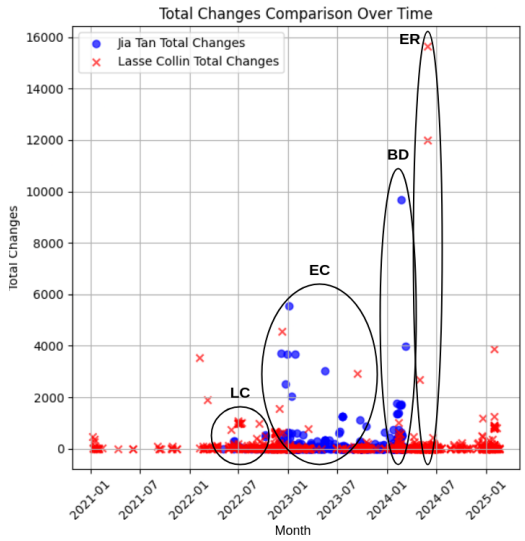

Fig. 1. Total Changes in Commits of authors Jia Tan and Lasse Collin

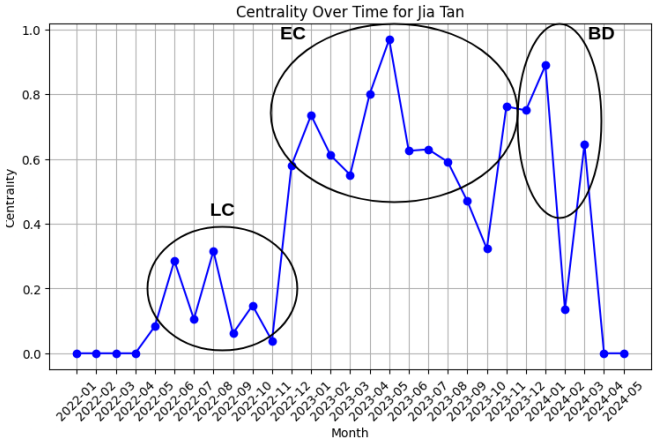

Fig. 2. Centrality of author Jia Tan

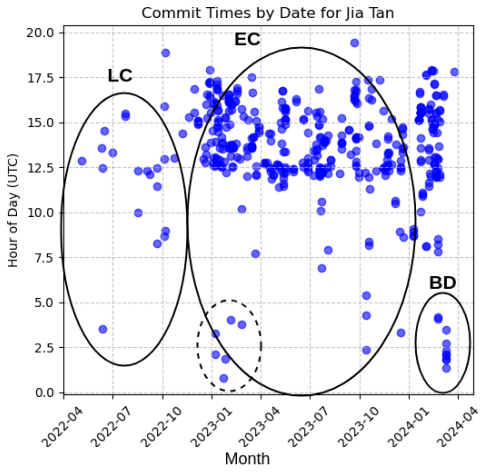

Fig. 3. Author Jia Tan commits plotted over months and time of day.

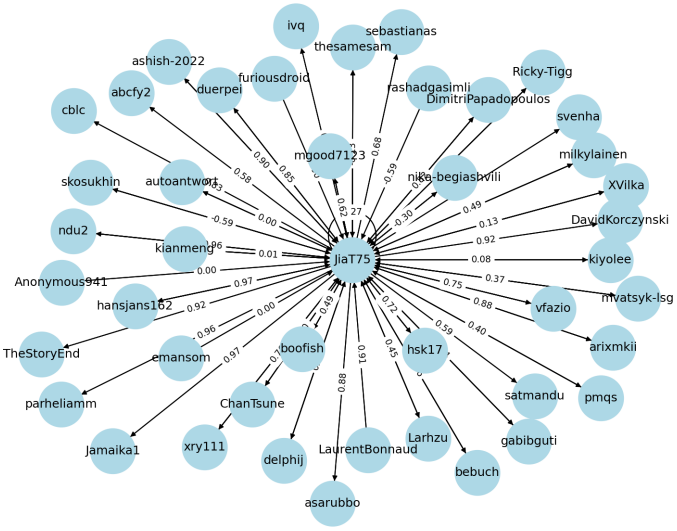

Fig. 4. Mean of the communication sentiment of author Jia Tan.

commit times (see later discussion of Fig. 3), *Jia Tan's* changes were both substantial and anomalous compared to the legitimate maintainer *Lasse Collin*.

To compare *Jia Tan's* impact upon the repository against all other developers, we built a network of the commits to each file and their authors. We then calculate node centrality (indicating the importance of an node (=author) in the network [39]) over time for *Jia Tan*, shown in Fig. 2. A higher value of centrality indicates that *Jia Tan's* influence upon the entire source code repository is high, with notable peaks at similar times to the malicious code entries [EC][BD].

The malicious contributions occurred at atypical times [35] which might indicate more than one person inserting the code due to working at a different times of day. Figure 3 plots *Jia Tan's* commits over time against the time of day. The cluster in the bottom right corner is the final malicious backdoor insertion [BD]. However, benign code had occasionally been submitted at atypical times of day (e.g. 2023-01), so alone, these results might not be seen as anomalous.

Further suspicious activities were related to the communications on the mailing list whereby *Lasse Collin* was subjected to a coordinated pressure campaign from multiple sock puppet accounts to relinquish control of the maintenance to *Jia Tan* [EC]. While data points in the mailing lists relating to these authors were limited, an analysis of the pressuring actors' sentiment, affinity for the suspicious actor *Jia Tan* and low interaction with the repository could indicate anomalous and untrustworthy actions. As an example, *Hans Jensen* entered the scene mid-2023 submitting a pair of patches that introduced the *ifunc* feature. These patches were reworked by *Lasse Collin* and then merged by *Jia Tan*. Beyond these interactions, and later filing a Debian bug requesting an update to XZ Utils v5.6.1, *Hans Jensen* does not exist anywhere else on the internet [34], [35].

Therefore, to analyse this communication, Fig. 4 presents a graph of *Jia Tan's* communication with other developers in the issues created in GitHub. The attribute on each edge is the mean value of the sentiment of that communication. Although this information alone is useful in identifying change in developer interactions, correlating it with change in components is necessary to identify any potential impact.

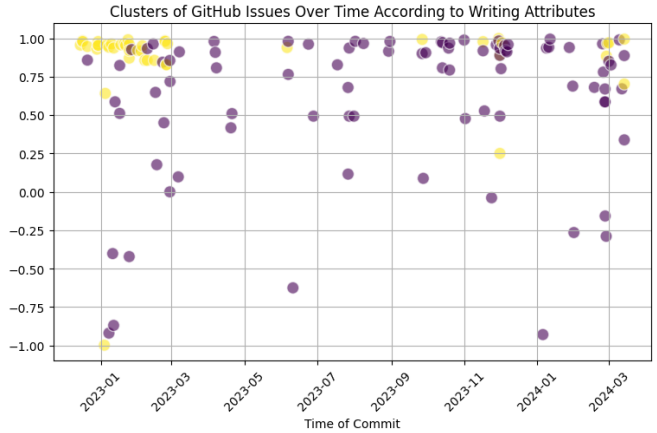

Fig. 5. Author Jia Tan issues' are clustered (yellow/purple) according to various writing features and plotted against the sentiment (y-axis) to identify if more than one person was using the account.

To further investigate whether *Jia Tan* was represented by more than one person, we examined various syntactic and lexical features of their writing in the GitHub's issues including: sentiment polarity, subjectivity, lexical richness, punctuation frequency, readability, and average sentence length using the Natural language toolkit [40] and TextBlob [41]. We then used the *k-means* algorithm (via sci-kit learn [42]) to cluster these writing features to identify whether there are several clusters of writing style, indicating that multiple authors where behind the username *Jia Tan*. Experimenting with different k values, i.e. the number of clusters, $k = 2$ separated clusters best, suggesting that two different persons might have been involved. Figure 5 plots for these two clusters over time against the sentiment of the issue text (y axis), i.e. the multi-dimensional values of the writing features are themselves are not depicted, only the two colours yellow and purple are used to distinguish the two assumed authors. The first cluster (yellow points), potentially indicating one author, has mainly a positive sentiment and is mainly found during escalation of control [EC], in particular in early 2023. The second cluster (purple points), potentially another author, takes over the majority of discussions starting from mid 2023. This cluster contains the majority of the negative sentiment while the pressure was being asserted (except for some outliers). The first author returns during the time the final backdoor is inserted in early 2024 [BD].

Examining these figures as a whole, it can be see that during the [EC] [BD] stages, *Jia Tan* has anomalous amount of lines changes in files (Fig. 1), increasing influence on the overall code base (Fig. 2), anomalous times of commits (Fig. 3), and potentially multiple authors with increasing negative sentiment in the issues that they created (Fig. 5). Therefore, this analysis shows that integrating technical and social data and analysing the trends might be used to identify anomalous developer behaviour which could indicate subversion and threats.

IV. Socio-Technical Adaptive Threat Detection

In this section, we propose *Socio-Technical Adaptive Threat Detection*. Figure 6 illustrates the approach by means of the

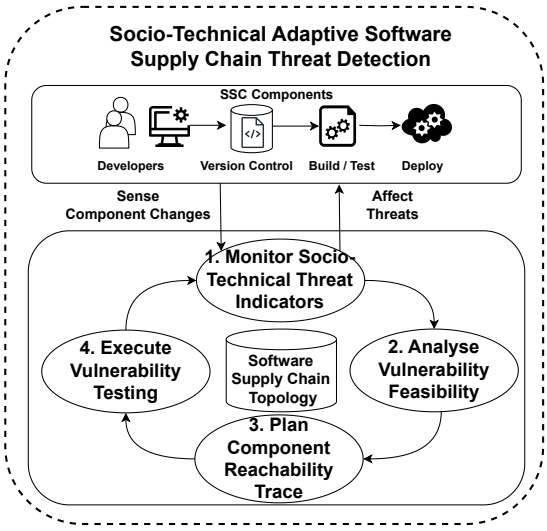

Fig. 6. SSC Threat Detection mapped on the MAPE-K framework

MAPE-K adaptive framework [33]. It will mine data sources (e.g. from version control systems, developer chat) to build and maintain socio-technical topologies of the supply chain (e.g. from code, tools, developer behaviour). The maintenance of these topologies will allow *monitoring* for change in social and technical components. Components where the dynamics between both social and technical components indicate a combined socio-technical vulnerability can then be *analysed* for the level of deviation from a previously known secure state (e.g. metrics such as lines of code and code churn). The threat to the SSC can then be identified through *planning* further *execution* of testing by tracing these components to determine further insecure states.

Building Socio-Technical Topology Models. Key to the approach is the ability to manage the structural complexity of the SSC. This requires *monitoring* the entire SSC for changes to prioritise components for vulnerability analysis. This includes modelling the *connectivity* between components, understanding the interplay between social and technical dimensions, and the *reachability* of vulnerabilities across the different components. To achieve this, we propose the use of multidimensional topologies – structural models – of the SSC [7]. To build these topologies, a variety of available data sources can be used to model them as graphs.

Formally, in this approach, a socio-technical topology model of a SSC is defined as a tuple $STT = (S, D, A, R, P, W, M, K, I)$ where:

- S is a set of source files S = $\{s_1, s_2, ..s_n\}$
- D a set of tuples representing dependency relationships between source files, $D \subseteq S \times S = \{(s_1, s_3), (s_1, s_2)...\}$.
- A is a set of authors who contribute to the software repository $A = \{a_1, a_2..., a_n\}$.
- R is a set of tuples defining relationships between authors $R \subseteq A \times A = \{(a_1, a_2), (a_1, a_3)\}$.
- P as a set of parameters that describe author relationships, such as communication frequency and sentiment, e.g. $P = \{frequency, sentiment, ...\}$

- W is a function assigning a weight to each relationship based on parameters $W : R \rightarrow \mathbb{R}^{|P|}$, e.g. $W(a_1, a_2) = (0.2, 0.9, ...)$
- M is a relation between maintainers and source files: $M \subseteq A \times S = \{(a_1, s_1), (a_1, s_2), ...\}$.
- K is a set of activities performed by maintainers while interacting with source files, e.g. $K = \{additions, delections, ...\}$
- I is a function defining the influence of maintainers on source files, based on their activities: $I : M \rightarrow \mathbb{N}^{|K|}$, e.g. $I(a_1, s_1) = (4, 12, ...)$.

1. Monitoring Socio-Technical Topologies for Threat Indicators. During this stage, data sources (e.g. version control systems, mailing lists) are mined to update the topology models over time. This timeline allows tracking change in the social and technical dimensions. Various metrics will be computed (e.g. code churn, network properties, communication frequency and sentiment, volatility of dependencies). Metrics changing together might indicate anomalous behaviour. An actor whose sentiment changes in tandem with changes in coding quality, commit time or overall influence on the code. The difference between $STTs$ at different times can be taken, e.g. $\Delta STT = STT(t_2) - STT(t_1)$. To identify suspect components they can be filtered, i.e. to select source files according to the developer influence and their changing relationship with other developers

$$S_R = \{s_i \mid \exists (a_1, a_2) \in R_2 - R_1 \text{ such that } s_i \text{ is involved in } (a_1, a_2) \text{ and } W(a_1, a_2) \geq W_{\text{threshold}}\} \quad (1)$$

and to filter according to their influence on a file:

$$S_I = \{s_i \mid \exists (a_j, s_i) \in I_2 - I_1 \text{ such that } I(a_j, s_i) \geq I_{\text{threshold}}\} \quad (2)$$

Then, to select components which fulfil both criteria: $S_{\text{selected}} = S_R \cap S_I$. $W_{\text{threshold}}$ and $I_{\text{threshold}}$ would need to be determined through monitoring the nominal behaviour of the system, as with any anomalous detection system.

2. Analysing Components for Vulnerability Feasibility. The filtered components identified previously will be combined with associated socio-technical threat indicators to analyse the feasibility of one or more vulnerability class existing including: 1) Design flaws such as through architectural model-checking and antipatterns, e.g. from Common Weakness Enumerators[1]. 2) Implementation bugs using static analysis and known vulnerable code, e.g. from OWASP TOP 10[2]. 3) Configuration errors through validation of insecure default settings. 4) Operation and Maintenance, e.g. through dependency auditing and CI/CD logs.

3. Planning Component Vulnerability Reachability Trace. If a feasible vulnerability is identified in one of the selected components in the previous stage, the *reachability* of that vulnerability must then be determined through tracing its impact upon downstream and upstream components in the SSC. The first stage can simply identify adjacent components where D_s^+ is the set of adjacent upstream components in the form $\{p', p\}$ and D_s^- the set of downstream components of the form $\{p, p'\}$.

4. Executing Vulnerability Testing. An analysis will need to test identified components, pruning elements of the dependability tree where the feasibility is low. This testing could take several forms, such as analysing the syntax and semantics of the code, e.g. through static or dynamic analysis, fuzz testing or machine learning approaches where the particular approach will need to be evaluated and selected according to features such as data availability.

V. Limitations and Research Vision

In this section, we present a discussion of the proposed approach and identify future research challenges to achieve it.

Adaptation. We argue that threat detection in SSCs must be adaptive to manage its high rate of change. While adaptative software frameworks are proven in many domains, SSCs pose challenges due to the heterogeneity and distributed nature of the components. The adaptation rate must consider this rate of change to be effective and the control functionality will need to accommodate the decentralised supply chain structure. Where engineering decentralised adaptation is an on-going research challenge [33].

Developer Behaviour Analysis. Developer activities as indicators for vulnerabilities have been investigated previously (Section II). However, these sources are large, varied and continuously evolving, e.g. new tools such as generative AI. To achieve the vision of this paper, we need to understand: *What types of developer behaviour can support identification of insecure states in SSC components? and How does the efficacy of these approaches vary between SSC components?* Eliciting the appropriate technique for each software repository will require context-aware methods.

Software Component Analysis. Analysing individual software components for vulnerabilities has seen considerable work in literature (Section II), although less so considering interdependency between components. New modelling approaches which apply metrics for vulnerability feasibility, suitable for comparison across diverse components are needed. Hence, to achieve this we seek to understand: *What are the patterns which can be used to identify sequences of insecure states in SSC components?* Identifying patterns of connected, potentially vulnerable components may support efficient identification of vulnerabilities without exhaustive search.

Evaluation of the Approach. Identifying SSC attacks is challenging due to sparse attack data, although this is increasing. In part, we mitigate this by identifying suspicious activities used to inform targeted vulnerability analysis. Regardless, we seek to understand: *How effectively does an adaptive approach support socio-technical threat detection in SSCs?* Comparing adaptive and non-adaptive approaches will require a controlled test bed covering the complete supply chain from design to deployment. The creation of such a test bed will strongly

[1]https://cwe.mitre.org/

[2]https://owasp.org/www-project-top-ten/

benefit SSC security research. Yet, while some open-source data is available, diverse social data and tool configurations will need to be developed.

VI. Conclusion

In this position paper, we outlined our research vision of a *Socio-Technical Adaptive SSC Threat Detection*. We motivated the need to build socio-technical topologies to understand the relationship between socio-technical dynamics and vulnerabilities through analysis of the data from the XZ Utils attack. We conclude with several challenges to the realisation of this approach, including investigating social and technical analysis techniques and the need for a SSC testbed to support research and evaluation efforts.

Acknowledgements

This project has received co-funding from the European Union's Digital Europe Programme under grant agreement no. 101127453 National Coordination Centre for Cybersecurity in Iceland and 101127307 Defend Iceland: Nationwide bug bounty platform.

References

[1] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "SoK: Analysis of software supply chain security by establishing secure design properties," in *Proc. 2022 ACM Workshop Soft. Supply Chain Offensive Res. and Ecosyst. Defenses*, 2022.

[2] Sonatype. (2024) 10th annual state of the software supply chain. Accessed: 2025-03-08. [Online]. Available: https://www.sonatype.com/state-of-the-software-supply-chain/introduction

[3] W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Secur. Priv.*, vol. 20, no. 2, 2022.

[4] R. Alkhadra, J. Abuzaid, M. AlShammari, and N. Mohammad, "Solar winds hack: In-depth analysis and countermeasures," in *2021 12th Int. Conf. Computing Commun. Netw. Technologies (ICCCNT)*. IEEE, 2021.

[5] S. Feng and M. Lubis, "Defense-in-depth security strategy in log4j vulnerability analysis," in *2022 Int. Conf. Adv. Data Science, E-learning Inf. Syst. (ICADEIS)*. IEEE, 2022.

[6] L. Williams *et al.*, "Research directions in software supply chain security," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, 2025.

[7] T. Welsh, F. Alrimawi, A. Farahani, D. Hassett, A. Zisman, and B. Nuseibeh, "Topology-aware adaptive inspection for fraud in I4.0 supply chains," *IEEE Trans. Ind. Inform.*, vol. 19, no. 4, 2022.

[8] P. Przymus and T. Durieux, "Wolves in the repository: A software engineering analysis of the xz utils supply chain attack," in *2025 IEEE/ACM 22nd Int. Conf. Min. Softw. Repositories (MSR)*. IEEE, 2025.

[9] T. Stalnaker *et al.*, "BOMS away! Inside the minds of stakeholders: A comprehensive study of bills of materials for software systems," in *Proc. 46th IEEE/ACM Int. Conf.Softw. Eng.*, 2024.

[10] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of attacks on open-source software supply chains," in *2023 IEEE Symposium Secur. Priv. (SP)*. IEEE, 2023.

[11] M. Lins, R. Mayrhofer, M. Roland, D. Hofer, and M. Schwaighofer, "On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from XZ," *ArXiv e-print 2404.08987*, 2024.

[12] E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia, and F. Palomba, "The secret life of software vulnerabilities: A large-scale empirical study," *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, 2022.

[13] I. Rauf *et al.*, "Influences of developers' perspectives on their engagement with security in code," in *Proc. 15th Int. Conf. Coop. Hum. Asp. Softw. Eng.*, 2022.

[14] M. Ivory, J. Towse, M. Sturdee, M. Levine, and B. Nuseibeh, "Recognizing the known unknowns; the interaction between reflective thinking and optimism for uncertainty among software developer's security perceptions," *Technol. Mind Behav.*, vol. 4, no. 3, 2023.

[15] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona, "Perceval: software project data at your will," in *Proc. 40th Int. Conf. Softw. Eng.: companion proceeedings*, 2018.

[16] D. M. German, B. Adams, and A. E. Hassan, "Continuously mining distributed version control systems: an empirical study of how Linux uses Git," *Empir. Softw. Eng.*, vol. 21, 2016.

[17] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 6, 2017.

[18] T. Lopez, T. Tun, A. Bandara, L. Mark, B. Nuseibeh, and H. Sharp, "An anatomy of security conversations in Stack Overflow," in *2019 IEEE/ACM 41st Int. Conf. on Softw. Eng.: Softw. Eng. in Soc. (ICSE-SEIS)*. IEEE, 2019.

[19] P. E. Carlson, "Engaging developers in open source software projects: harnessing social and technical data mining to improve software development," Ph.D. dissertation, Iowa State University, 2015.

[20] M. Kuutila, M. V. Mãntylã, and M. Claes, "Chat activity is a better predictor than chat sentiment on software developers productivity," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng.*, 2020.

[21] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Comput. Surv.*, vol. 50, no. 4, Aug. 2017. [Online]. Available: https://doi.org/10.1145/3092566

[22] M. Masum *et al.*, "Quantum machine learning for software supply chain attacks: How far can we go?" in *2022 IEEE 46th Annual Comput., Softw. Appl. Conf. (COMPSAC)*. IEEE, 2022.

[23] B. Al Sabbagh and S. Kowalski, "A socio-technical framework for threat modeling a software supply chain," *IEEE Secur. Priv.*, vol. 13, no. 4, 2015.

[24] G. Baxter and I. Sommerville, "Socio-technical systems: From design methods to systems engineering," *Interact. Comput.*, vol. 23, no. 1, 2011.

[25] W. Syafitri, Z. Shukur, U. A. Mokhtar, R. Sulaiman, and M. A. Ibrahim, "Social engineering attacks prevention: A systematic literature review," *IEEE access*, vol. 10, 2022.

[26] T. Caldwell, "Prepare to fail: creating an incident management plan," *Comput. Fraud Secur.*, vol. 2012, no. 11, 2012.

[27] N. Gcaza and R. Von Solms, "Cybersecurity culture: an ill-defined problem," in *Inf. Secur. Edu. Glob. Digital Soc.: 10th IFIP World Conf. Proc. 10*. Springer, 2017.

[28] M. Goerzen, E. A. Watkins, and G. Lim, "Entanglements and exploits: Sociotechnical security as an analytic framework," in *9th USENIX Workshop Free Open Commun Internet (FOCI 19)*, 2019.

[29] A. Vespignani, "Modelling dynamical processes in complex socio-technical systems," *Nat. Phys.*, vol. 8, no. 1, 2012.

[30] E. Estrada, *The structure of complex networks: theory and applications*. American Chemical Society, 2012.

[31] G. Tóth *et al.*, "Inequality is rising where social network segregation interacts with urban topology," *Nat. Commun.*, vol. 12, no. 1, 2021.

[32] F. Alrimawi, L. Pasquale, and B. Nuseibeh, "On the automated management of security incidents in smart spaces," *IEEE Access*, vol. 7, 2019.

[33] F. Quin, D. Weyns, and O. Gheibi, "Decentralized self-adaptive systems: A mapping study," in *2021 Int. Symposium Softw. Eng. Adaptive Self-Managing Systems (SEAMS)*. IEEE, 2021.

[34] R. Cox, "XZ timeline," 2024, accessed: 2025-03-07. [Online]. Available: https://research.swtch.com/xz-timeline

[35] Kaspersky Securelist, "XZ backdoor story, part 2: Social engineering," 2024, accessed: 2025-03-07. [Online]. Available: https://securelist.com/xz-backdoor-story-part-2-social-engineering/112476

[36] MITRE, "CVE-2024-3094," 2024, accessed: 2025-03-07. [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2024-3094

[37] The Tukaani Project. (2025) XZ Utils. GitHub repository https://github.com/tukaani-project/xz. Accessed: 2025-03-11.

[38] ——. (2025) XZ Utils mailing list archives. https://www.mail-archive.com/xz-devel@tukaani.org/. Accessed: 2025-03-11.

[39] S. P. Borgatti, "Centrality and network flow," *Soc. Netw.*, vol. 27, no. 1, 2005.

[40] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc., 2009.

[41] S. Loria. (2018) TextBlob documentation. Release 0.15. [Online]. Available: https://textblob.readthedocs.io

[42] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, 2011.