LAGOON Final Report / Demonstration, Including OSS Contributor Ascendency

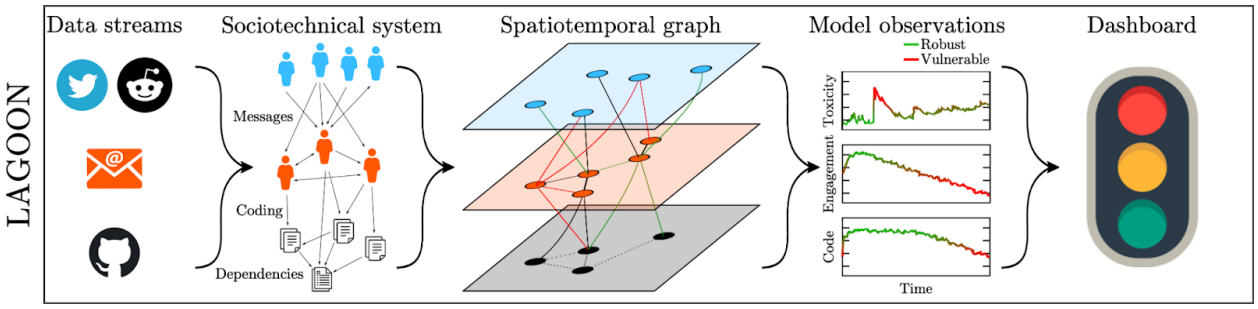
Galois and the University of Vermont

In compliance with the fourth milestone of DARPA SocialCyber:

Case Studies: Demonstration of integrated analysis tools which characterize patterns of OSS contributor ascendency to the effective authority on free software projects, measured by both trusted critical code contributions and taking architectural decisions into consideration.	Phase I Report and Demonstration

Comprehensive Summary

DoD applications continue to rely on Open Source Software (OSS) for economic and labor force purposes. Ensuring the supply-chain integrity of these dependencies is crucial for the security of downstream applications. The LAGOON project has resulted in a brand-new, open source platform which helps analysts understand OSS communities from a social-oriented security perspective. Focusing on the observable artifacts produced within these communities, LAGOON provides a full suite of tools for ingesting different kinds of data, fusing it into a unified, sociotechnical and spatiotemporal graph, and then leveraging Machine Learning (ML)-enabled capabilities to help predict and prevent future attacks against OSS software that has real-world effects on downstream projects, all of which is shown in the figure below. The platform is currently designed to be an efficient tool for Observe, Orient, Decide, Act (OODA) loop scenarios, though a continuous integration version could be developed in the future.



Specific LAGOON results and capabilities funded by SocialCyber

- **DoD-relevant use cases** for LAGOON or other SocialCyber platforms. We have identified multiple different, real-world software supply-chain attacks which could be detected through the data and algorithms available through LAGOON. Specifically, these

attacks stem from either a central contributor becoming dissatisfied with their project or its user base, or from a low-contribution community member who attempts to assert the requisite authority in order to get a malicious patch accepted. Therefore, LAGOON's focus on both quantifying and predicting the wellbeing of core maintainers via toxicity, as well as characterizing new contributors, is aligned with detecting real-world attacks.

- **Open source products**, the LAGOON and OCEAN software. These allow for the ingest of multiple data sources, fusing them into one sociotechnical temporal graph, and running various analyses on the result. Each new community ingested only takes a few hours of human time. The LAGOON repository includes a ready-to-go, 1.5GB database of the CPython and LXML communities, including 39699 users (with Have I Been Pwned data for 36424 of them), 207120 git commits, 799019 OCEAN mailing list messages, 31500 GitHub pull requests, 50133 GitHub conversations, 570 Python Enhancement Proposals (PEPs), and 19737 files from the Python + LXML communities. The ingest of more data happens in layered batches, allowing for incremental or partial updates, where the order of data ingestion is often irrelevant to the correctness of the system.
- **Reports on contributor and organizational dynamics** within OSS communities. which demonstrate the qualities of different community members, including contributor ascendency and power on both technical and social levels. Ability to temporally characterize a contributor's involvement and authority over the community. Allowed us to identify "trending topic" vs "broad topic" contributors. Ability to look at organizational factors of the Python community.
- **Investigation of LXML CVEs** using the LAGOON platform. Looking at seven CVEs related to the popular LXML library, LAGOON's database was searched for evidence of these CVEs being tied to deliberate attacks. While we were able to identify the contributors and commits tied to many of these vulnerabilities, we found them to be the results of legitimate oversight instead of coordinated attacks. Similar processes were conducted on CPython CVEs to investigate the feasibility of connecting them to specific files within the project's source code; 16 CVEs resulting from specific files were found.
- **Publications highlighting the SocialCyber program**. Three publications were submitted as part of these efforts; two accepted to the 2022 Mining Software Repositories conference (LAGOON and OCEAN, both accepted), and one to the 2022 Sunbelt conference (pending notification). The publications cover both the LAGOON and OCEAN software, as well the use of these tools to study the networks within OSS development communities.
- **Understanding of the predictive power of aggregate toxicity versus constellations of toxicity** within the Python community. Specifically, we note that aggregating toxicity in a neighborhood around a contributor or topic of interest is insufficient for making predictive claims about a contributor disengaging from the project, topping out at 3% improvement compared to a naive guess. On the other hand, ML methods can leverage constellations of toxicity within the sociotechnical graph to account for variations in contributors disengaging 11.7% better than a naive guess.
- **OSS-community-aware toxicity detector** fine-tuned from the BERT language model. This model was trained on data from Wikipedia edits, which helped to tune it toward polysemy-aware modalities of toxicity, as one would expect in the more professional

environment of OSS development. On a held-out set from the Wikipedia edits, this model achieved 90% accuracy. However, on a balanced set hand-labeled from the CPython OSS community, the BERT model achieved 69% accuracy, whereas a previous wordlist-based approach only achieved 40% accuracy.

- **Social network analysis of OSS integrating toxicity classification** exploring the social effects of toxicity in the collaboration networks of the Python community. We find that users producing messages classified as toxic tend to be more active and central than users with no messages classified as toxic, but have fewer collaborators per project.
- **Future research objectives** which could be used to significantly improve on results from the SocialCyber program. Expanding the scope of analysis and focusing on foundational advances to the science of dealing with large, complicated sociotechnical systems are both crucial. Expanding the scope includes both considering multiple OSS communities in tandem, as modern software tends to have a supply chain that includes multiple OSS dependencies, as well as looking at a large array of perspectives on that data from those communities. That is, the SocialCyber program both provides a way to look at OSS communities holistically and demonstrates that these communities are rich macrocosms of human interaction. Understanding this complex fabric in isolation – that is, only looking at artifacts from OSS communities – is less likely to be successful than understanding it in the context of other social and behavioral research. For foundational advances in science, we point to the rampant overfitting that happens with a rich graph of interactions as LAGOON provides, and how mechanisms of dealing with that overfitting are necessary for results that generalize well. Other topics are provided too.

DoD-relevant use cases

Examples of historic attacks which LAGOON seeks to detect include: the pushing of malicious commits to the Linux kernel, the unpublishing of LeftPad, the deliberate corruption of Faker.js by its author, and the unauthorized publishing of malicious versions of UA-Parser.js. All of these attacks are DoD-relevant: with an increasing reliance on OSS, the attack surface accessible by these software supply-chain attacks just keeps growing. Procedural attempts at addressing these vulnerabilities exist, such as dependency freezing or semantic versioning, but each has their own issues. While freezing dependencies might prevent pulling newly pushed malicious code, as proposed in this article about UA-Parser.js, it also prevents necessary security patches for zero days from being pulled into production deployments. On the other hand, semantic versioning was proposed as a means of ensuring interoperability and availability of security patches, yet it is fundamentally driven by human processes, and its patchwork adoption makes it not a reliable means of preventing supply-chain dependency attacks. One might make the claim that the DoD could simply eschew OSS; however, the cost of that would likely be prohibitive if adopted on a large scale. To quote the Army on a Modular Open Systems Approach, which relies on integration of OSS: “While it’s hard to quantify the ROI (savings, cost avoidance), it is clear that MOSA is the right strategy in many cases and should always be considered. Based on projected savings in the area of avionics, convergence of hardware and

software to a MOSA construct is expected to yield an ROI in the hundreds of millions of dollars." [https://apps.dtic.mil/sti/pdfs/AD1040338.pdf]

While the technical substance of these attacks varies, the majority either stem from a central contributor becoming dissatisfied with their project or its user base, or from a low-contribution community member who attempts to assert the requisite authority in order to get a malicious patch accepted[2]. By definition, low-contribution community members have little data with which to discern their intent, and on significant OSS projects, often cannot modify the primary source code repository without a core maintainer's sign-off. OSS projects therefore rely almost wholly on the wellbeing and alertness of their core maintainers, making most of these supply-chain vulnerabilities largely the result of social processes. This is not the same as arguing that the only community members worth understanding are the core maintainers; the LAGOON project was designed specifically to capture the effects of this general public on the core maintainers, and to help identify concerted efforts which might be coordinated, emotional denial of service attacks against an OSS project's maintainers. That is, toxicity and other interpersonal dimensions tied to project communications affect the psychology of an OSS project's maintainers, which in turn can lead to a maintainer becoming disgruntled or otherwise unable to safeguard the project.

LAGOON is therefore a general purpose security platform for projects that rely on OSS. It is designed to be pointed at one or more OSS projects, and multiple data sources which provide artifacts that give information on the community defined by these projects. It then provides a suite of tools and reports that allow analysts to inspect a variety of socially-focused, security-relevant qualities, such as sub-team associations, the ebb and flow of a contributor's influence (ascendency), likelihood of a core maintainer becoming disengaged, or gathering a list of maintainers responsible for a specific file or feature set.

Add new layers / what differentiates LAGOON from similar platforms. Novelty of developed platform. Talk about database flexibility – allowing broad experiments rather than deep. Pull from our MSR 2022 paper.

Open source products

The open source OCEAN and LAGOON repositories allow users to ingest and explore sociotechnical artifacts resulting from OSS projects, including mailing list messages, source code repositories, GitHub pull requests, and Python Enhancement Proposals (PEPs; a key organizational mechanism of the Python community). These artifacts give insight into members of OSS communities, and let analysts inspect these members' power within OSS organizations as well as those members' social pressures and responses to those pressures. To provide mailing list capabilities, LAGOON leverages the Open-source Complex Ecosystem And Networks (OCEAN) partnership between the University of Vermont and Google Open Source. To provide GitHub pull request capabilities, LAGOON leverages TwoSix labs' scraper, which

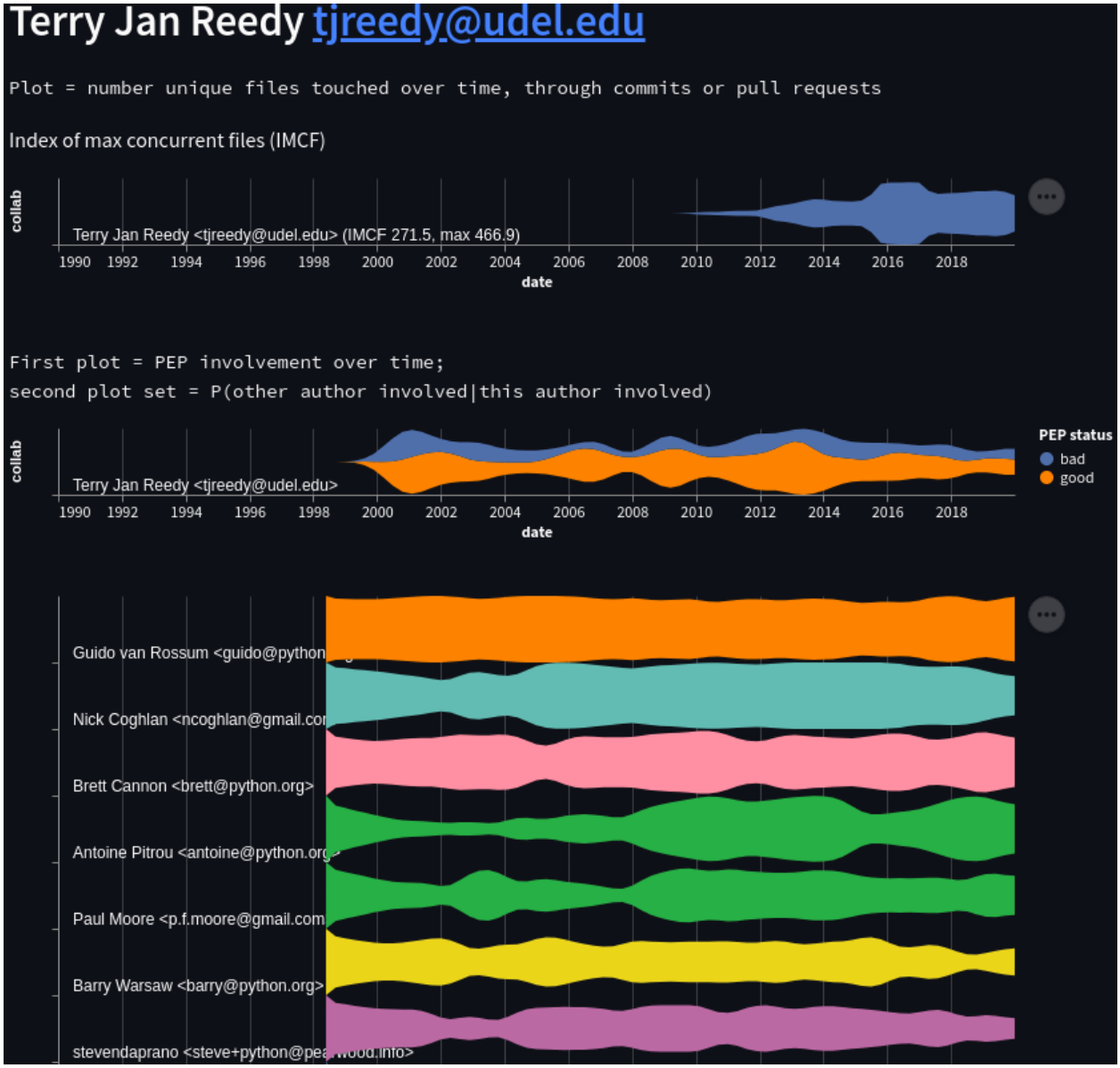
[2] The exception to this dichotomy – UA-Parser.js – could be caught by adding a check for discrepancies between deployment code (as available from NPM) with underlying source code (as available on GitHub).

was also created as part of the SocialCyber program.

As a case study, LAGOON focused on CPython, the most widely used implementation of the Python programming language. To demonstrate agility, we also added LXML, the rank #67 most downloaded package from Python's PyPI package repository, which has accrued seven CVEs over the years. Adding the LXML source code repository and GitHub pull requests to the database only took about 3 hours of human effort to identify data sources and queue them up to be pulled into the platform.

Reports on contributor and organizational dynamics

LAGOON looks at individual contributors from two perspectives: the breadth of their technical involvement and the extent of their organizational influence. Technical involvement is measured through the Index of Maximum Concurrent Files (IMCF), which measures the attack surface available to a contributor if they were malicious. Organizational influence is investigated via Python Enhancement Proposals, the core mechanism by which architectural decisions for Python are made. Putting these two quantities together gives a comprehensive view of a contributor's involvement in the OSS project:



Both of these are detailed in below sections, notably “Contributor ascendency” and “Organizational structures amongst Python contributors.”

Investigation of LXML CVEs

An ideal outcome from SocialCyber would be specific examples of attacks against OSS. Generally, while investigating the CPython implementation of Python, we found the community to be healthy, with files being actively maintained by numerous contributors. No evidence of deliberate attacks was found in this community.

To increase our chances of finding attacks, we decided to look at the LXML library, which is downloaded 33.6 million times per month and has 7 CVEs. Additionally, this library is poorly funded, an environment which might permit malicious commits more readily than the base

Python OSS. Using the LAGOON database and software, we investigated these CVEs, working backward through the source code repository and tooling to find the commits which allowed for these vulnerabilities. Our investigation found these CVEs to derive from legitimate oversights rather than deliberate efforts.

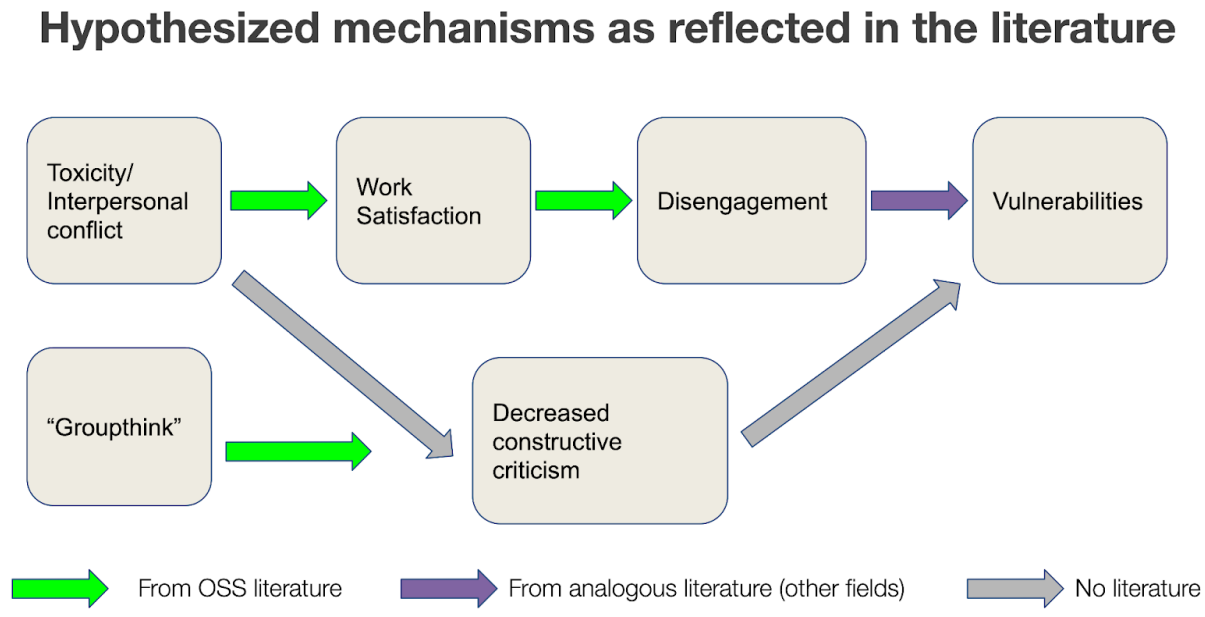
The process of investigating CVEs is detailed in the “Since the M6 Milestone” section “Investigation of LXML CVEs.”

Additionally, linkage from CPython CVEs to specific source code files was investigated using the LAGOON platform. In total, 16 CVEs were found to have descriptions that contained enough information to link them to specific source code files. Future work could use this information to identify if these CVEs were the results of deliberate efforts. This effort is detailed in the “Since the M6 Milestone” section “Linking CPython CVEs to Source Code Files.”

Publications highlighting the SocialCyber program

Three publications were submitted: two to the 2022 Mining Software Repositories (MSR) conference, and one to the 2022 Sunbelt conference. Pre-prints are available for the papers submitted to MSR on LAGOON and OCEAN.

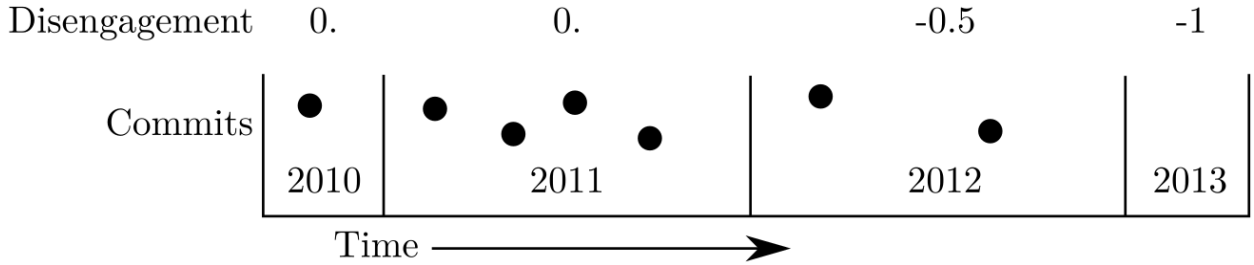
A literature survey was also conducted which places LAGOON’s results in context of other researchers’ efforts; notably, there is evidence that in volunteer-driven efforts, like OSS, community toxicity leads to disengagement, which leads to declines in product quality. For OSS, a decline in quality implies the potential for uncaught vulnerabilities. The LAGOON platform allows for researchers to explore these hypotheses in greater detail specifically as they relate to OSS projects.



This is covered in more detail in the “Since the M6 Milestone” section, under the “Publications and academic merit” subsection.

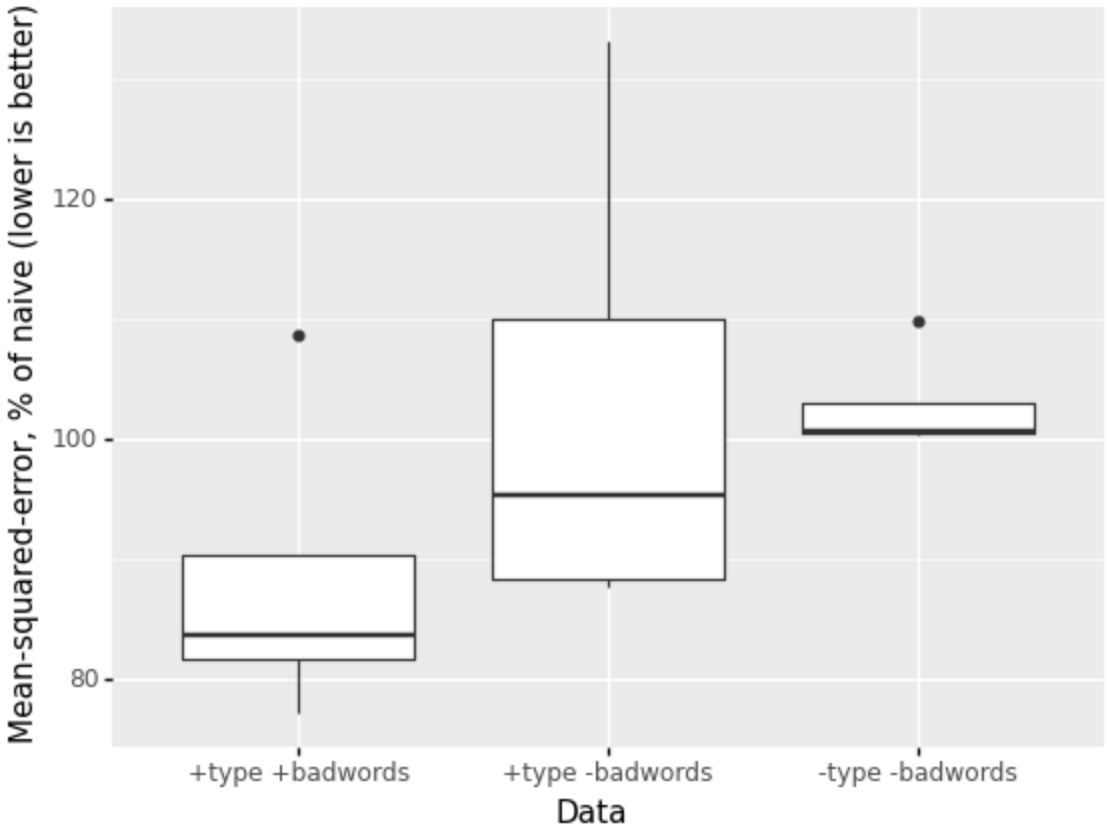
Understanding of the predictive power of aggregate toxicity versus constellations of toxicity

Disengagement was chosen as an example metric that we might automatically compute and predict for contributors. As noted previously, disengagement leads to a reduction in project quality, and actively reduces the number of maintainers with enough familiarity to ensure security.



Since there is not much published work in this space, we discuss results in terms of percent improvement over a naive baseline. Initial results on predicting disengagement using information from the Python community looked only at aggregates. Even well-known ML methods, like the Graph Convolutional Network (GCN), effectively only look at the aggregate of a local neighborhood, and then re-combine this aggregate using standard weight-then-activate layers. Our best models that leveraged simple aggregates were able to predict disengagement 3% better than the naive baseline.

On the other hand, recent advances in ML have yielded concepts like the transformer’s multi-headed attention block. By adapting this block to work with a GCN-like architecture, we produced an architecture that was able to leverage the full constellation of data around a contributor. That is, rather than only computing based on simple mean aggregates, the transformer-enhanced GCN could look at multiple views of the data simultaneously, granting attention to whichever aspects improve computation the most. This model was able to predict disengagement 11.7% better than the naive baseline. We also demonstrated that this model was able to make significant use of toxicity information (“badwords” below) in the service of predicting disengagement:



OSS-community-aware toxicity detector

Sentiment classification in professional contexts is a difficult problem. For example, toxicity filters from the wider internet often flag “black” as used more often in toxic contexts than well-meaning ones. However, the Python community uses a piece of software called “black” for no-nonsense, automatic formatting of code. These overloaded words, or polysemy, illustrate that each community or subcommunity has different aspects which they consider to be toxic.

To deal with this, a BERT-based toxicity classifier was fine-tuned from a labeled dataset of Wikipedia edits. This scored 90% accurate on a balanced set from the Wikipedia data. We then hand-labeled all of the commits from the Python community, and constructed a balanced dataset of 50 toxic and 50 non-toxic examples specifically from our target community. The fine-tuned toxicity classifier scored 69% accuracy on this dataset, demonstrating the difficulty of transferring sentiment analysis results between communities. However, this is compared to 40% accuracy on the Python toxicity dataset from our previous, bad-word-list-based approach.

As an example of a sentence which looks toxic to a word filter, because of the word “hate”, but is correctly identified as non-toxic by the BERT classifier, consider this sentence:

> [NOT TOXIC] “I hate to call them 'keyword' arguments when they aren't (check with the\nkeyword module: it will confirm they aren't keywords!-).”

More details may be found in the “Verifying transfer of domain-specific toxicity from Wikipedia to OSS” section later in this report.

Social network analysis of OSS integrating toxicity classification

To reduce the complexity of the sociotechnical system studied by LAGOON, we projected OSS projects on co-committing networks. This network representation acts as a proxy for a collaboration network as it captures user interactions through the co-editing of shared files. Doing so then opened the toolbox of social network analysis as a means to investigate how toxicity affects collaborations and productivity.

We found that accounts sending messages classified as toxic tend to slow down productivity and collaborations on specific pieces of software (toxic commit messages). Perhaps as a consequence, we also observed that these accounts generally gain less collaborations per activity over time and, more directly, work on files with less collaborators. All together, this analysis demonstrates the mostly speculated impact of toxicity in OSS and highlights a potential pathway for targeted social attacks to slow down productivity and compromise OSS communities.

Future research objectives

The following would be appropriate avenues for continued exploration, and to develop tools that significantly improve on results from the SocialCyber program:

- Research on code / community health metrics that directly lead to CVEs in an OSS project would be immensely helpful for follow on work. There is currently no unified understanding of the qualities of a project that allow for the intrusion of malicious code. The SocialCyber program was a significant step toward being more aware of the interrelation between a project’s social environment and the security of its technical artifacts, and yet this domain needs significantly more study.
- A wider scope of projects and communities being investigated by each team would help test out the completeness of this approach for detecting supply-chain vulnerabilities in OSS. Most modern projects do not rely on a single OSS dependency. In fact, often tens or hundreds of packages are pulled in, either directly or as dependencies of dependencies. In these real-world scenarios, being able to cope with the scale of many OSS communities becomes vital for assembling a reliable dashboard to help with assessing the overall security of a downstream project. We note that the inclusion of LXML into the LAGOON database was a step in this direction.
- The basic science surrounding graph NNs, particularly as it relates to overfitting mitigation, needs to be improved. Since graphical relations are composed of unique constellations of data (as opposed to, e.g., uniform images), it is easy for ML methods to memorize the data rather than compute a generalizing function. This problem is

demonstrated by results from Graph Convolutional Networks (GCNs), where a 2-hop neighborhood is the maximum beneficial function to compute. Large, complex social graphs, like those found in SocialCyber, need advances in the fundamental science underlying learning on graphical data in order to leverage all available information.

- The integration of a wide set of influence cues would allow analysts to better tie observations in OSS communities to wide psychology or sociology literature. Due to the breadth of data available from OSS communities, pulling in external studies that lend credibility to certain interpretations of the data might be a great way to cut through some of the overfitting endemic to the SocialCyber problem. For example, in collaboration with the University of Florida, one might integrate the Potentiam dataset to differentiate between contributors posturing with loss framing to persuade the community away from potentially more secure architectures, versus those who speak more objectively.
- A more generalizable mechanism is needed for automatically identifying architectural decisions and authoritative figures in those decisions. Our analyses used the PEP mechanism which was unique to the Python community. While these analyses were highly insightful, they are specific to Python, and a more generalized mechanism would likely need to rely more heavily on natural language processing to cover formal decision processes.
- A better understanding and ability to compute different subcommunities of interaction amongst contributors. Our results found that some teams interact more toxically within themselves, yet that particular team was no more or less productive than other teams. This is an example of a subculture, or group of people within the larger OSS community that interact differently with one another. Being able to identify and handle these subcommunities will result in a more accurate understanding of the overall community and its rich internal dynamics.
- Better language modeling tools, which can more accurately predict sentiment or intent within natural language messages *in technical domains*, would greatly boost the confidence of presented results. While we were able to utilize the state-of-the-art BERT model to achieve an accuracy of 69% at detecting toxicity within the CPython OSS community, there clearly is significant progress to be made before reaching 95% confidence. While the current accuracy levels are enough for cursory analysis, operationalizing results would benefit from more accuracy in the underlying methods.
- Integration of static analysis-based tools with the social-centric tools. The social-centric approaches that fall under SocialCyber are necessary, and perhaps even principal to the health of an OSS project given how subtly malicious effects may be implemented. Nonetheless, as an additional prior that helps focus analyst attention, being able to leverage static analyses to help identify potentially malicious code, and combining that with social awareness of the contributors, would be a more holistic approach. This would be a significant effort, however, and would likely be closer to productization than standard prototype development. We note that recent improvements in ML, such as Code2vec, might also be of use in identifying code abstractions that correlate to specific maintainer mentalities or intents.

Since the M6 milestone

Since the M6 milestone, Galois and UVM have refined the LAGOON platform. LAGOON has been used to derive results for multiple academic publications that emphasize the necessity of the SocialCyber program. Additionally, we have provided evidence that detecting malicious contributors before they cause harm to OSS projects is feasible. We have ingested the LXML community as well, to demonstrate the agility of our toolchain and ability to deal with numerous dependencies of a downstream project. New reports and metrics have been integrated into LAGOON to help better characterize the ascendency of community members for both social and technical involvement in the projects being investigated. These improvements are summarized through the key achievements of the last three months:

- (Galois) **Demonstrated contributor ascendency** to effective authorities on the CPython and LXML projects with two approaches: 1) **tracking architectural decisions** via the PEP collaboration mechanism, and 2) **looking at a contributor's potential attack surface** via the Index of Maximum Concurrent Files (IMCF).
- (Galois) **Submitted a paper to the MSR 2022 Data and Tool Showcase Track** documenting the LAGOON platform and its capabilities (arXiv:2201.11657).
- (UVM) **Submitted two publications**: a **paper to the MSR 2022 Data and Tool Showcase** Track, which focuses on the OCEAN dataset, as well as a **talk to the Sunbelt 2022 Conference**, which leverages LAGOON to examine, "how toxic interpersonal signals impact and reflect collaboration and prestige in the open source Python language development community."
- (Galois) **Ingested the `lxml` OSS project** alongside CPython to demonstrate LAGOON's ability to cope with communities comprised of multiple OSS projects. to demonstrate ease of using LAGOON tooling on larger, multi-project communities.
- (Galois) **Investigation of LXML CVEs** using the LAGOON platform. Looking at seven CVEs related to the popular LXML library, LAGOON's database was searched for evidence of these CVEs being tied to deliberate attacks. While we were able to identify the contributors and commits tied to many of these vulnerabilities, we found them to be the results of legitimate oversight instead of coordinated attacks.
- (UVM) **Linkage from CPython CVEs to source code files** using the LAGOON platform. In total, 16 CVEs were found to have descriptions that contained enough information to link them to specific source code files. Future work could use this information to identify if these CVEs were the results of deliberate efforts.
- (Galois) **Collaborated with RiverLoop / TwoSix to add GitHub pull request data** into the LAGOON platform; this was made straightforward due to the powerful modular architecture of LAGOON. Between CPython and LXML, this resulted in the addition of 31500 GitHub pull requests and 50133 GitHub conversations to the database.
- (UVM) Verified **transfer of learning from fine-tuned toxicity detection** on a Wikipedia-derived dataset to the OSS-specific mailing list for the Python community. Accuracy of 90% on the Wikipedia-derived dataset became 69% on OSS-specific data.
- (UVM) Leveraging domain-specific toxicity filter, **re-ran network analyses**. Discovered that **users producing messages classified as toxic tend to be very active but less collaborative** than users with no messages classified as toxic.

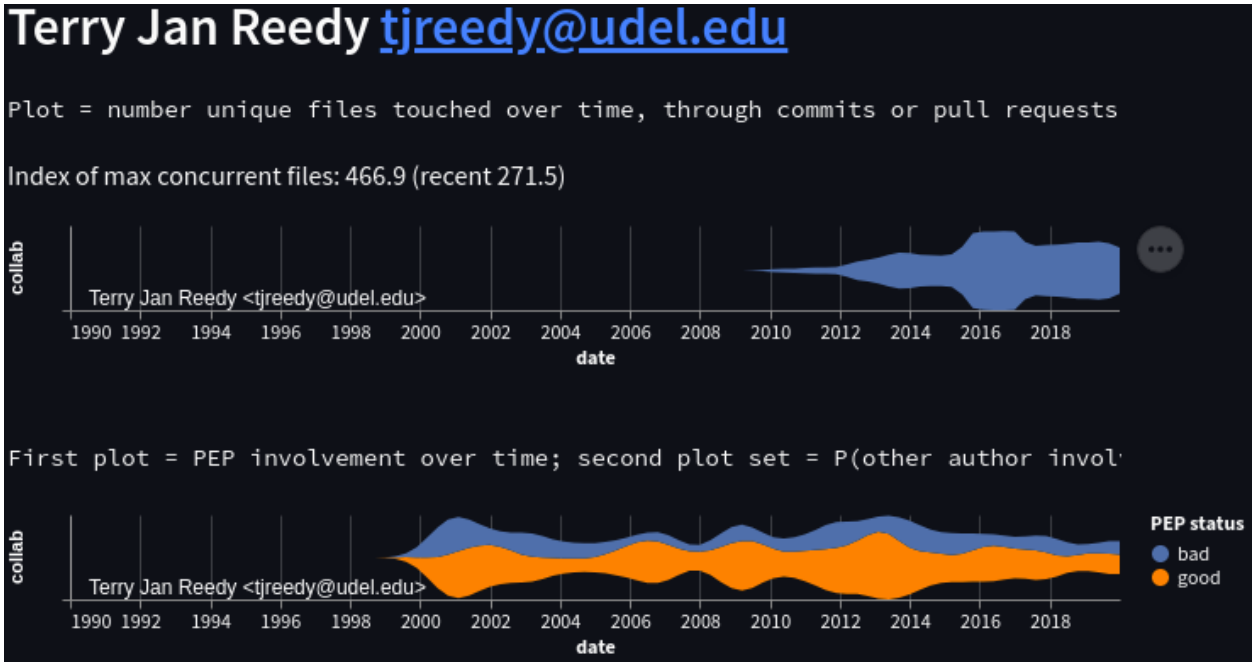
- (UVM) Literature review demonstrating that **disengagement reduces the quality of volunteer-driven efforts**. Since OSS is predominantly a volunteer effort, and its security is part of its quality, this demonstrates that **disengagement might allow for malicious contributors to commit weird machines**.
- (Galois) **Identified potentially overlooked source code regions** with low or anomalous maintainer patterns.
- (Galois) Leveraged ML to identify malicious users by predicting their status in the **Have I Been Pwned** (HIBP) database.
- (Galois) Leveraged UVM's BERT-based classifier to **re-analyze PEPs and contributor disengagement using new toxicity features**.
- (Galois) **Numerous improvements to LAGOON's UI**, including better data accessibility, significantly faster entity search, and clickable GitHub links to ease integration with other data sources.

Contributor ascendency

As of M6, we had demonstrated LAGOON's ability to assess both: 1) a contributor's effective authority with respect to architectural decisions, and 2) the change in that authority over the project's lifetime. That work was presented in the section on "Organizational structures amongst Python collaborations," which has been reiterated in the final sections of this report. Briefly, it involved looking at Python Enhancement Proposals (PEPs), the decision unit of the Python programming language. PEPs are formal proposals written by one or more community members, which go through a rigorous approvals process. The process is reasonably discriminative, with 59% of PEPs being adopted by the community on average.

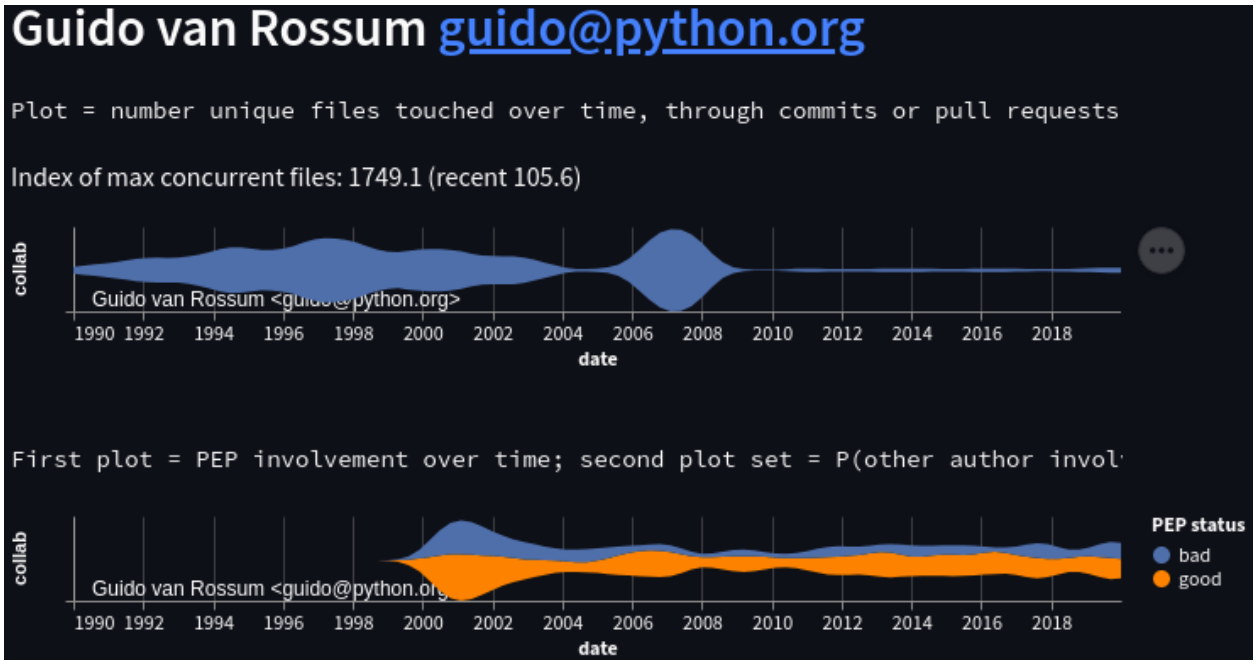
To enhance this report to account for critical code contributions for M9, we've conceptualized and implemented an Index of Maximum Concurrent Files (IMCF). The IMCF measures the area of potential attack surface for a contributor if they were malicious; this is derived from the breadth of a contributor's impact on the code base. The index is created by looking at all commits a community member is responsible for, including those attached to pull requests[3]. Each file touched by these commits then gets its own list of times at which that specific file was touched by this user. These touches are then convolved with a Gaussian kernel with a standard deviation of half a year, and then divided by the maximum height of such a Gaussian for a single file and clamped to a maximum of 1. The result is then summed across all files the user has ever touched, which produces a temporal graph like the first one below:

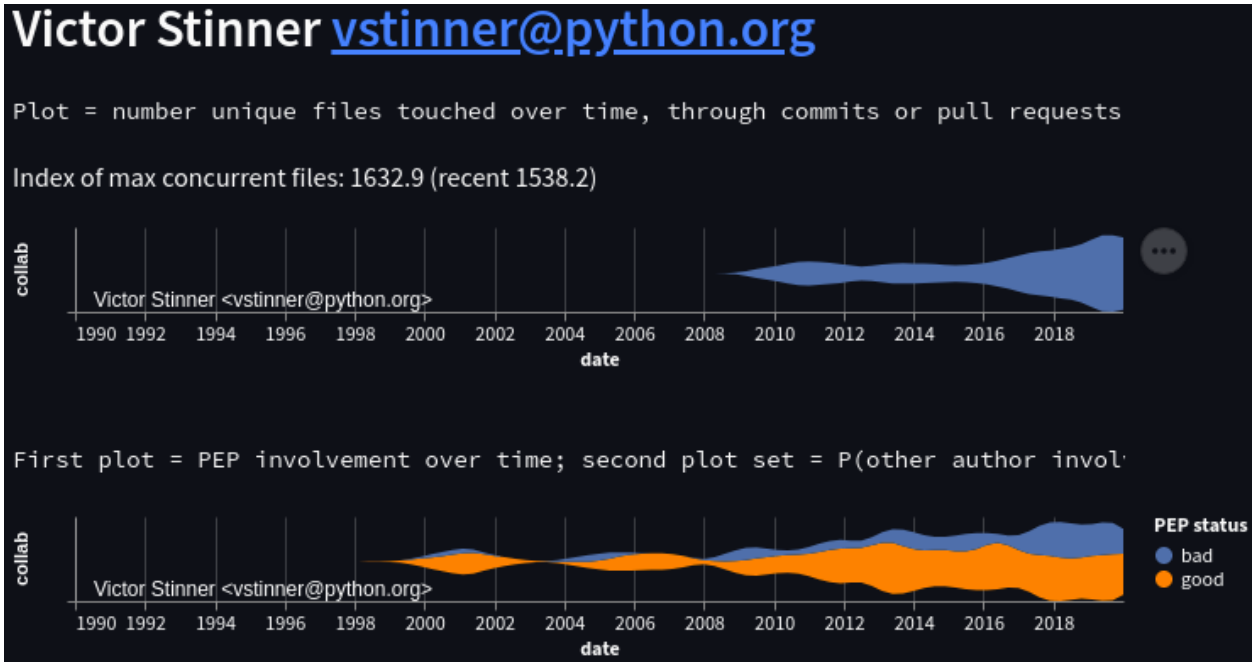
[3] Due to issue #3 in TwoSix's GitHub scraper, many pull requests were not correctly associated with commits in the git repository at time of program conclusion. As a result, the included plots primarily address influence resulting from direct code contributions rather than review activities.



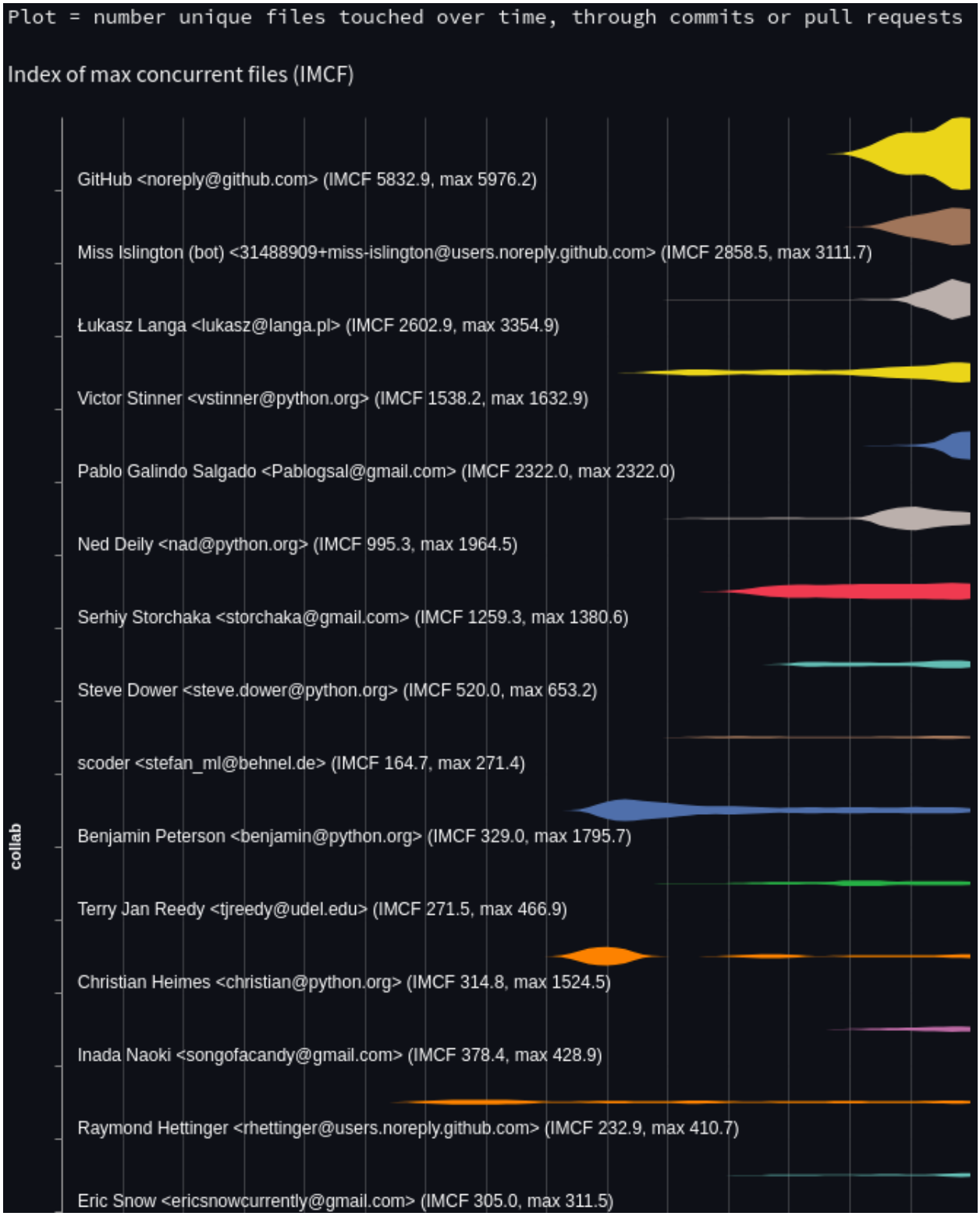
When paired with reports on organizational power, such as the above PEP involvement, this provides a comprehensive view of each contributor's ascendency in terms of both technical and social capital within the given OSS community.

For reference, here are a few more key contributors; notably, Guido formally stepped down as Python's "Benevolent Dictator for Life" (BDFL) in July 2018, though still seems reasonably involved with the project; Victor Stinner is a core contributor who's organizational connections were analyzed as part of the "Organizational structures amongst Python collaborations" section:





More broadly, here is a characterization of the contributors with the highest IMCF within the last three years:



Notably, this points at another potential vulnerability in modern codebases: automated contributions from bots like GitHub or Miss Islington, the latter of which is Python-specific. Generally, these bots are trusted and fulfill either a security or developer quality-of-life function; however, these bots themselves are composed of code which might be targeted by malicious

commits. Furthermore, as indicated in the figure above, commits from these bots can touch many different files simultaneously, making it difficult for human reviewers to verify the security of the entire patch.

In summary, the IMCF provides a means of looking at the scope of each contributor's critical code contributions. This quantity corresponds to the attack surface available to that contributor if they were malicious. Importantly, changes in IMCF over time demonstrate a contributor's ascendency in terms of ability to directly inject malicious code. The IMCF can be coupled with knowledge of a contributor's involvement with organizational decisions, such as PEPs, in order to also gauge their ability to enact architectural changes that would increase the efficacy of malicious code. These two numbers can both be plotted over time, and together provide a reasonably comprehensive view of influence on any OSS project. While these numbers cannot be used to directly make claims regarding the intent of a contributor, they could be used to determine the importance of that contributor's intent to the overall project's health and security.

Investigation of LXML CVEs

Common Vulnerability Exposures (CVEs) are a common means of tracking security issues in popular software projects and libraries. In an attempt to find coordinated attacks against software within the larger OSS community, we looked at the LXML library. LXML ranks #67 on PyPI's most downloaded packages list as of March 1st 2022, with 33.6 million downloads per month, and has 7 CVEs according to MITRE. Despite large adoption, LXML's funding is small, amounting to just under 3400 EUR/year on average for 2019 and 2020. Therefore, LXML would be a reasonable target for a coordinated effort.

In an attempt to find such a coordinated effort, we used LAGOON tooling to look at the various LXML CVEs and work backward. The CVEs are detailed here:

CVE-2021-43818	lxml is a library for processing XML and HTML in the Python language. Prior to version 4.6.5, the HTML Cleaner in lxml.html lets certain crafted script content pass through, as well as script content in SVG files embedded using data URIs. Users that employ the HTML cleaner in a security relevant context should upgrade to lxml 4.6.5 to receive a patch. There are no known workarounds available.
CVE-2021-33511	Plone though 5.2.4 allows SSRF via the lxml parser. This affects Diazo themes, Dexterity TTW schemas, and modeleditors in plone.app.theming, plone.app.dexterity, and plone.supermodel.

CVE-2021-28957	An XSS vulnerability was discovered in python-lxml's clean module versions before 4.6.3. When disabling the safe_attrs_only and forms arguments, the Cleaner class does not remove the formaction attribute allowing for JS to bypass the sanitizer. A remote attacker could exploit this flaw to run arbitrary JS code on users who interact with incorrectly sanitized HTML. This issue is patched in lxml 4.6.3.
CVE-2020-27783	A XSS vulnerability was discovered in python-lxml's clean module. The module's parser didn't properly imitate browsers, which caused different behaviors between the sanitizer and the user's page. A remote attacker could exploit this flaw to run arbitrary HTML/JS code.
CVE-2020-27197	** DISPUTED ** TAXII libtaxii through 1.1.117, as used in EclecticIQ OpenTAXII through 0.2.0 and other products, allows SSRF via an initial http:// substring to the parse method, even when the no_network setting is used for the XML parser. NOTE: the vendor points out that the parse method "wraps the lxml library" and that this may be an issue to "raise ... to the lxml group."
CVE-2018-19787	An issue was discovered in lxml before 4.2.5. lxml/html/clean.py in the lxml.html.clean module does not remove javascript: URLs that use escaping, allowing a remote attacker to conduct XSS attacks, as demonstrated by "j a v a s c r i p t:" in Internet Explorer. This is a similar issue to CVE-2014-3146.
CVE-2014-3146	Incomplete blacklist vulnerability in the lxml.html.clean module in lxml before 3.3.5 allows remote attackers to conduct cross-site scripting (XSS) attacks via control characters in the link scheme to the clean_html function.

Commits pertaining to recent CVEs from this list and external to the LXML project may be found by running the following in the LAGOON CLI:

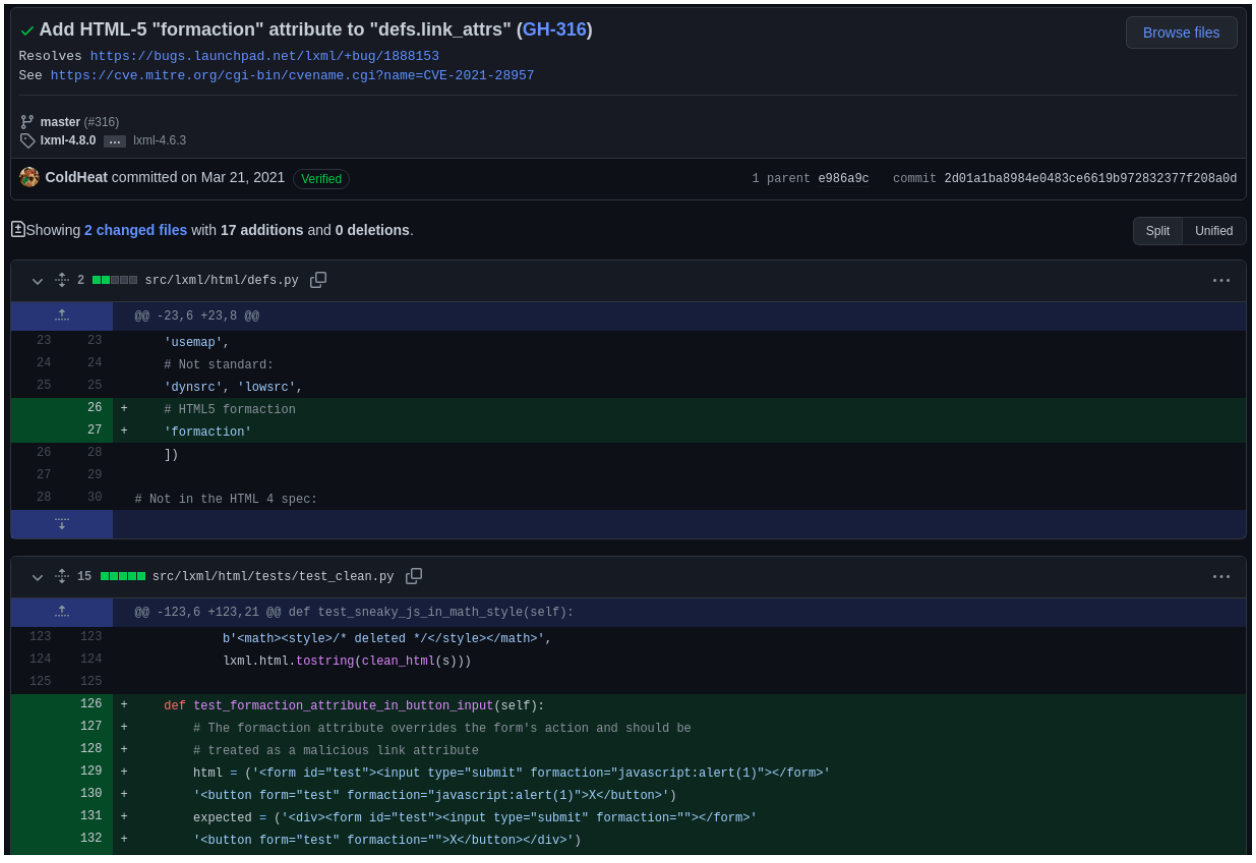
```
>>
    sess().query(sch.FusedEntity).where(sch.FusedEntity.attrs['message'].astext.op('~*')('cve-20')).count()
```

This reveals 144 commits related to any 2000-or-later CVEs; if we filter purely for the above list, which are LXML specific, we find precisely one commit, the details of which can be viewed in LAGOON's UI:

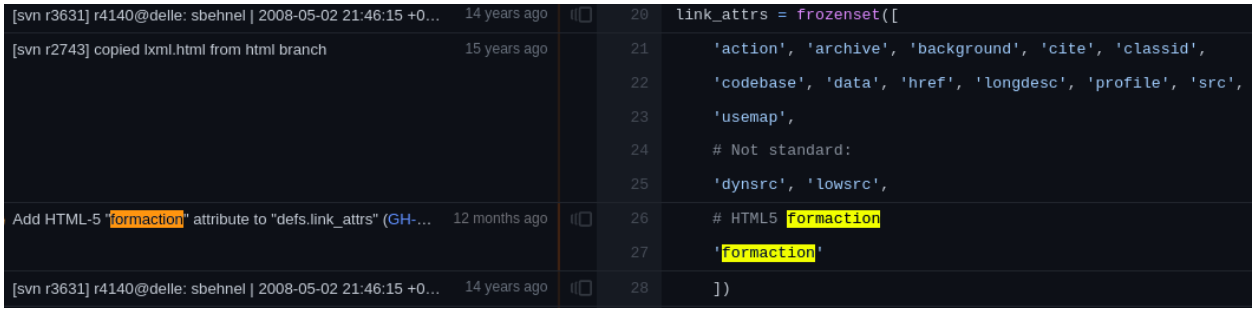
<lagoon.db.schema_fused.FusedEntity 17460650: EntityTypeEnum.git_commit 2d01a1ba89> FOCUS

time	1616335389
message	Add HTML-5 "formaction" attribute to "defs.link_attrs" (GH-316) Resolves https://bugs.launchpad.net/lxml/+bug/1888153 See https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-28957
commit_sha	2d01a1ba8984e0483ce6619b972832377f208a0d
obs_as_src	<lagoon.db.schema_fused.FusedObservation 17467373: (ObservationTypeEnum.modified@2021-03-21, 17460650, 17466562)>
obs_as_src	<lagoon.db.schema_fused.FusedObservation 17467374: (ObservationTypeEnum.modified@2021-03-21, 17460650, 17466537)>
obs_as_dst	<lagoon.db.schema_fused.FusedObservation 17467371: (ObservationTypeEnum.created@2021-03-21, 17466370, 17460650)>
obs_as_dst	<lagoon.db.schema_fused.FusedObservation 17467372: (ObservationTypeEnum.committed@2021-03-21, 17466351, 17460650)>
fusion	17460650 [58 ingest-git-github.com/lxml/lxml]

We can use LAGOON's GitHub integration to click on the "commit_sha" link, revealing that the change automatically strips the "formaction" element in order to prevent arbitrary javascript from being run on users' machines:



In this case, it would be very difficult to prove that whoever made the sanitizing list omitted `formaction` deliberately. We could use `git blame` to find out who that was:



Which reveals that code as 14 years old, and as being committed by the original maintainer of LXML, Stefan Behnel.

We could investigate the intent of the committer who fixed the CVE, Kevin Chung / ColdHeat via the LAGOON UI. In this case, the contributor only has the one commit, which demonstrably makes the project more secure by resolving a CVE.

This whole process took about 5-10 minutes for the CVE. For other CVEs which were not explicitly mentioned in commit messages, a few of them could be linked to specific commits via MITRE's "References" section, which occasionally includes links to specific commits on GitHub.

None of the CVEs inspected in relation to the LXML project demonstrated evidence of a coordinated attack, and instead appeared to be instances of legitimate oversight.

Linking CPython CVEs to Source Code Files

We also investigated CVEs within the core CPython library. To match CVEs with the appropriate file(s) in our database, we downloaded all CVEs ascribed to the core Python library from https://www.cvedetails.com/vulnerability-list/vendor_id-10210/Python.html. The format governing the submission of CVEs does not include a field which uniquely identifies which files in an open source repository contribute to the CVE. The only field of the CVE which potentially contains information about relevant files is the description. So we searched these descriptions for common phrases that indicate that one of the words in the phrase is a file name, and matched these with filenames in the database. We then manually checked these and matched records with clear indications of filename. We found that the majority of the examples in this cvedetails.com database under Python were actually third party libraries incorrectly ascribed to CPython such as Pillow and urllib3. We managed to match 16 CVEs to files in our database using these methods. These CVEs are shown below.

CVE ID	Description	files_affected	file_id
CVE-2019-16056	An issue was discovered in Python through 2.7.16, 3.x through 3.5.7, 3.6.x through 3.6.9, and 3.7.x through 3.7.4. The email module wrongly parses email addresses that contain multiple @ characters. An application that uses the email module and implements some kind of checks on the From/To headers of a message could be tricked into accepting an email address that should be denied. An attack may be the same as in	_parseaddr.py | Lib/urllib2.py	133102 | 141890

	CVE-2019-11340; however, this CVE applies to Python more generally.		
CVE-2021-3177	Python 3.x through 3.9.1 has a buffer overflow in PyCArg_repr in _ctypes/callproc.c, which may lead to remote code execution in certain Python applications that accept floating-point numbers as untrusted input, as demonstrated by a 1e300 argument to c_double.from_param. This occurs because sprintf is used unsafely.	Modules/_ctypes/callproc.c>	133191
CVE-2019-20907	In Lib/tarfile.py in Python through 3.8.3, an attacker is able to craft a TAR archive leading to an infinite loop when opened by tarfile.open, because _proc_pax lacks header validation.	Lib/tarfile.py	133253
CVE-2021-3426	There's a flaw in Python 3's pydoc. A local or adjacent attacker who discovers or is able to convince another local or adjacent user to start a pydoc server could access the server and use it to disclose sensitive information belonging to the other user that they would not normally be able to access. The highest risk of this flaw is to data confidentiality. This flaw affects Python versions before 3.8.9, Python versions before 3.9.3 and Python versions before 3.10.0a7.	Lib/pydoc.py	133547
CVE-2020-8492	Python 2.7 through 2.7.17, 3.5 through 3.5.9, 3.6 through 3.6.10, 3.7 through 3.7.6, and 3.8 through 3.8.1 allows an HTTP server to conduct Regular Expression Denial of Service (ReDoS) attacks against a client because of urllib.request.AbstractBasicAuthHandler catastrophic backtracking.	Lib/urllib/request.py	133637
CVE-2022-0391	A flaw was found in Python, specifically within the urllib.parse module. This module helps break Uniform Resource Locator (URL) strings into components. The issue involves how the urlparse method does not sanitize input and allows characters like '\r' and '\n' in the URL path. This flaw allows an attacker to input a crafted URL, leading to injection attacks. This flaw affects Python versions prior to 3.10.0b1, 3.9.5, 3.8.11, 3.7.11 and 3.6.14.	Lib/urllib/parse.py	133857
CVE-2021-23336	The package python/cpython from 0 and before 3.6.13, from 3.7.0 and before 3.7.10, from 3.8.0 and before 3.8.8, from 3.9.0 and before 3.9.2 are vulnerable to Web Cache Poisoning via urllib.parse.parse_qsl and urllib.parse.parse_qs by using a vector called parameter cloaking. When the attacker can separate query parameters using a semicolon (;), they can cause a difference in the interpretation of the request between the proxy (running with default configuration) and the server. This can result in malicious requests being cached as completely safe ones, as the proxy would usually not see the semicolon as a separator, and therefore would not include it in a cache key of an unkeyed parameter.	Lib/urllib/parse.py	133857
CVE-2019-9636	Python 2.7.x through 2.7.16 and 3.x through 3.7.2 is affected by: Improper Handling of Unicode Encoding (with an incorrect netloc) during NFKC normalization. The impact is: Information disclosure (credentials, cookies, etc. that are cached against a given hostname). The components are: urllib.parse.urlsplit, urllib.parse.urlparse. The attack vector is: A specially crafted URL could be incorrectly parsed to locate cookies or authentication data and send that information to a different host than when parsed correctly. This is fixed in: v2.7.17, v2.7.17rc1, v2.7.18, v2.7.18rc1; v3.5.10, v3.5.10rc1, v3.5.7, v3.5.8, v3.5.8rc1, v3.5.8rc2, v3.5.9; v3.6.10, v3.6.10rc1, v3.6.11, v3.6.11rc1, v3.6.12, v3.6.9, v3.6.9rc1; v3.7.3, v3.7.3rc1, v3.7.4, v3.7.4rc1, v3.7.4rc2, v3.7.5, v3.7.5rc1, v3.7.6, v3.7.6rc1, v3.7.7, v3.7.7rc1, v3.7.8, v3.7.8rc1, v3.7.9.	Lib/urllib/parse.py	133857
CVE-2021-29921	In Python before 3,9,5, the ipaddress library mishandles leading zero characters in the octets of an IP address string. This (in some situations) allows attackers to bypass access control that is based	Lib/ipaddress.py	133906

	on IP addresses.		
CVE-2020-14422	Lib/ipaddress.py in Python through 3.8.3 improperly computes hash values in the IPv4Interface and IPv6Interface classes, which might allow a remote attacker to cause a denial of service if an application is affected by the performance of a dictionary containing IPv4Interface or IPv6Interface objects, and this attacker can cause many dictionary entries to be created. This is fixed in: v3.5.10, v3.5.10rc1; v3.6.12; v3.7.9; v3.8.4, v3.8.4rc1, v3.8.5, v3.8.6, v3.8.6rc1; v3.9.0, v3.9.0b4, v3.9.0b5, v3.9.0rc1, v3.9.0rc2.	Lib/ipaddress.py	133906
CVE-2020-26116	http.client in Python 3.x before 3.5.10, 3.6.x before 3.6.12, 3.7.x before 3.7.9, and 3.8.x before 3.8.5 allows CRLF injection if the attacker controls the HTTP request method, as demonstrated by inserting CR and LF control characters in the first argument of HTTPConnection.request.	Lib/test/multibytecodec_support.py	134444
CVE-2019-9674	Lib/zipfile.py in Python through 3.7.2 allows remote attackers to cause a denial of service (resource consumption) via a ZIP bomb.	Lib/zipfile.py	134751
CVE-2019-16935	The documentation XML-RPC server in Python through 2.7.16, 3.x through 3.6.9, and 3.7.x through 3.7.4 has XSS via the server_title field. This occurs in Lib/DocXMLRPCServer.py in Python 2.x, and in Lib/xmlrpc/server.py in Python 3.x. If set_server_title is called with untrusted input, arbitrary JavaScript can be delivered to clients that visit the http URL for this server.	Lib/DocXMLRPCServer.py | Lib/xmlrpc/server.py	141985| 133642
CVE-2019-9948	urllib in Python 2.x through 2.7.16 supports the local_file: scheme, which makes it easier for remote attackers to bypass protection mechanisms that blacklist file: URIs, as demonstrated by triggering a urllib.urlopen('local_file:///etc/passwd') call.	Lib/urlopen.py	146834
CVE-2018-20852	http.cookiejar.DefaultPolicy.domain_return_ok in Lib/http/cookiejar.py in Python before 3.7.3 does not correctly validate the domain: it can be tricked into sending existing cookies to the wrong server. An attacker may abuse this flaw by using a server with a hostname that has another valid hostname as a suffix (e.g., pythonicexample.com to steal cookies for example.com). When a program uses http.cookiejar.DefaultPolicy and tries to do an HTTP connection to an attacker-controlled server, existing cookies can be leaked to the attacker. This affects 2.x through 2.7.16, 3.x before 3.4.10, 3.5.x before 3.5.7, 3.6.x before 3.6.9, and 3.7.x before 3.7.3.	Lib/http/cookiejar.py	136105

Publications and academic merit

Three publications were submitted: two to the 2022 Mining Software Repositories conference, and one to the 2022 Sunbelt conference. The abstracts and details are reproduced below.

Additionally, a literature survey was conducted to uncover ties between LAGOON's primary metric of toxicity and security-centric concerns within OSS software projects.

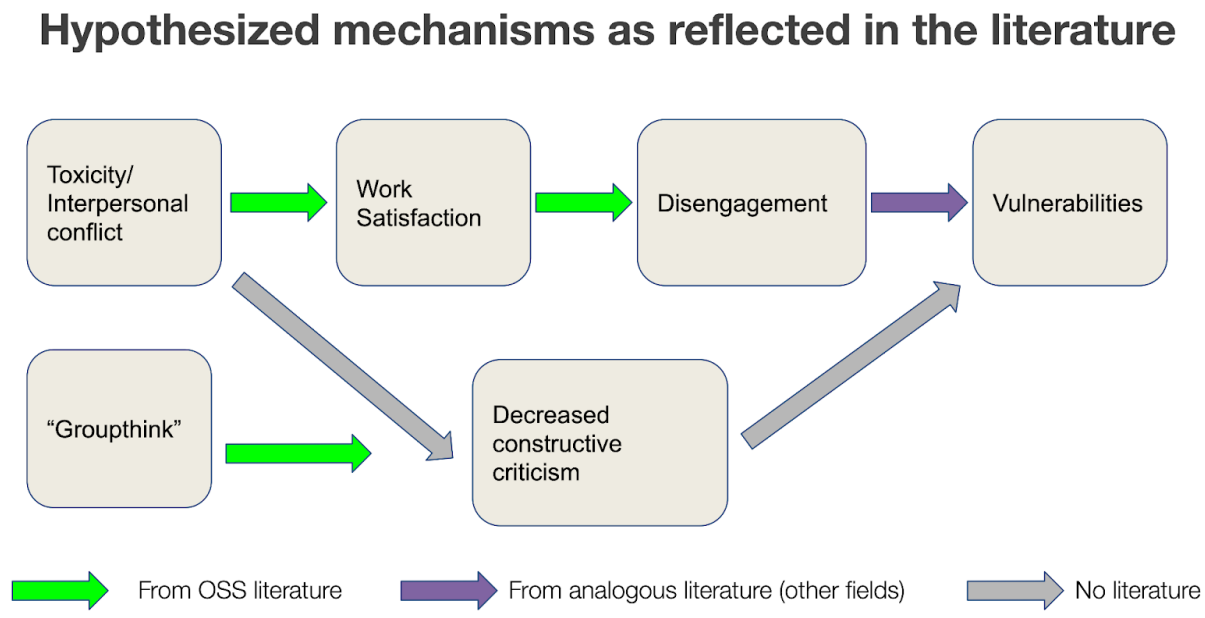
Given that open source projects are by definition, volunteer endeavors, the factors that influence sustained volunteerism are frequently investigated. These investigations often assume that disengagement is worth preventing to ensure high project quality [Fang & Neufeld, 2009; Lu et al., 2022, Miller et al., 2019]. Additionally, these efforts suggest the value of LAGOON and OCEAN, which enable future large-scale studies and might be used to find actionable results

that “could significantly increase the sustainability of open source ecosystems” [Miller et al., 2019].

However, while the outcomes of volunteer disengagement in OSS projects have been studied in terms of project failure [Coelho & Valente, 2017], and individual recognition of performance [Roberts et al., 2006], the impact that disengagement has on the quality (and by extension the security) of OSS projects which do not fail is under-researched. Knowledge from other domains [Zhou et al., 2008] suggests that satisfaction is positively related to product quality. Thus, we would expect project dissatisfaction resulting from toxicity to negatively impact code quality, and by extension, security.

Additionally, Colazo [2010] and Peng et al. [2013] demonstrated that interpersonal dynamics can affect code quality through mechanisms besides disengagement. Colazo theorized that one of these mechanisms was “groupthink”, whereby isolated teams developed homogeneous sets of beliefs which hindered creativity and critical thinking. Toxicity could potentially act in a related manner, where toxic individuals bully others into accepting their set of beliefs or become themselves immune to productive critical analysis by simply being too wearisome to critique (his is illustrated in the figure below).

While early detection systems for open source software vulnerabilities do exist [Iorga et al., 2020; Iorga et al. 2021], to the best of our knowledge, none of these utilize features based on interpersonal contributor dynamics such as those the LAGOON tooling is designed to generate, despite the fact that research suggests interpersonal dynamics affect product quality as measured by metrics other than security[]. More research using the LAGOON tooling will lead to better answers to these questions, as we are able to link community toxicity to vulnerabilities either discovered through LAGOON or published via CVEs.



Citations from Literature Review:

Fang, Y., & Neufeld, D. (2009). Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, 25(4), 9-50.

Lu, X., Ai, W., Chen, Z., Cao, Y., & Mei, Q. (2022). Emojis predict dropouts of remote workers: An empirical study of emoji usage on GitHub. *PloS one*, 17(1), e0261262.

Coelho, J., & Valente, M. T. (2017, August). Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (pp. 186-196).

Roberts, Jeffrey A.; Hann, Il-Horn; Slaughter, Sandra A. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science*, 52(7), 984–999. doi:10.1287/mnsc.1060.0554

Iorga, D., Corlătescu, D., Grigorescu, O., Săndescu, C., Dascălu, M., & Rughiniş, R. (2020, December). Early detection of vulnerabilities from news websites using machine learning models. In *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)* (pp. 1-6). IEEE.

Iorga, D., Corlatescu, D. G., Grigorescu, O., Sandescu, C., Dascalu, M., & Rughinis, R. (2021, May). Yggdrasil—Early Detection of Cybernetic Vulnerabilities from Twitter. In 2021 *23rd International Conference on Control Systems and Computer Science (CSCS)* (pp. 463-468). IEEE.

Colazo, J. A. (2010). Collaboration structure and performance in new software development: findings from the study of open source projects. *International Journal of Innovation Management*, 14(05), 735-758.

Peng, G., Wan, Y., & Woodlock, P. (2013). Network ties and the success of open source software development. *The Journal of Strategic Information Systems*, 22(4), 269-281.

Zhou, K. Z., Li, J. J., Zhou, N., & Su, C. (2008). Market orientation, job satisfaction, product quality, and firm performance: evidence from China. *Strategic management journal*, 29(9), 985-1000.

LAGOON: An Analysis Tool for Open Source Communities

Accepted at MSR 2022 Data and Tool Showcase Track
arXiv available: arXiv:2201.11657

Abstract: This paper presents LAGOON -- an open source platform for understanding the complex ecosystems of Open Source Software (OSS) communities. The platform currently utilizes spatiotemporal graphs to store and investigate the artifacts produced by these communities, and help analysts identify bad actors who might compromise an OSS project's security. LAGOON provides ingest of artifacts from several common sources, including source code repositories, issue trackers, mailing lists and scraping content from project websites. Ingestion utilizes a modular architecture, which supports incremental updates from data sources and provides a generic identity fusion process that can recognize the same community members across disparate accounts. A user interface is provided for visualization and exploration of an OSS project's complete sociotechnical graph. Scripts are provided for applying machine learning to identify patterns within the data. While current focus is on the identification of bad actors in the Python community, the platform's reusability makes it easily extensible with new data and analyses, paving the way for LAGOON to become a comprehensive means of assessing various OSS-based projects and their communities.

The OCEAN mailing list data set: Network analysis spanning mailing lists and code repositories

Accepted at MSR 2022 Data and Tool Showcase Track
Preprint available: https://bagrow.com/pdf/warrick2022-mailing-v1.pdf

Abstract: Communication surrounding the development of an open source project largely occurs outside the software repository itself. Historically, large communities often used a collection of mailing lists to discuss the different aspects of their projects. Multimodal tool use, with software development and communication happening on different channels, complicates the study of open source projects as a sociotechnical system. Here, we combine and standardize mailing lists of the Python community, resulting in 954,287 messages from 1995 to the present. We share all scraping and cleaning code to facilitate reproduction of this work, as well as smaller datasets for the Golang (122,721 messages), Angular (20,041 messages) and Node.js (12,514 messages) communities. To showcase the usefulness of these data, we focus on the CPython repository and merge the technical layer (which GitHub account works on what file and with whom) with the social layer (messages from unique email addresses) by identifying 33% of GitHub contributors in the mailing list data. We then explore correlations between the valence of social messaging and the structure of the collaboration network. We discuss how these data provide a laboratory to test theories from standard organizational science in large open source projects.

Open is not always welcoming: Examining how toxic interpersonal signals impact and reflect collaboration and prestige in the open source Python language development community

Submitted to Sunbelt 2022

Abstract: According to most indices, Python is the most popular programming language worldwide. Combining decades of commit records with community mailing-lists, we have created a temporally fine-grained, multi-perspective view of the open source community developing the Python language. Our goal is to explain contributor quality dynamics, such as disengagement or accession, through analysis of interpersonal signals in the mailing-list data (e.g. toxicity). We do this by examining structural changes in the co-commit interaction networks, in reference to categorical changes like appointment of topical experts.

We find that a small group of (community-officiated) experts make up over half of commits and messages. In the co-commit network this translates to higher mean degree and betweenness centrality compared to non-experts, by factors of four and ten, respectively. Despite this, we find that community-appointed experts are like average contributors in many respects. Compared to non-experts, experts had no significant differences in levels of ambient toxicity among collaborators nor in levels of toxic word use themselves. However, when first designated experts, contributors experienced a decline in ambient toxicity of collaborators. These preliminary results are surprising considering the association between power and negative ties noted by prior research, suggesting differences between centrally-organized institutions and self-organized open source communities.

Adding new data sources with minimal human effort

The LAGOON platform was designed with a number of features which make it easy to ingest new or updated information. A few key architectural decisions make this possible:

1) A layered, batch-oriented ingestion system, which makes it easy to replace or incrementally update old information.
2) An integrated entity fusion system to merge identities of e.g. users which may occur in multiple different data sources, or even multiple times within a single data source. This makes the order in which data was ingested irrelevant to the correctness of the system.
3) A flexible database model: data within LAGOON is categorized as either an entity or an observation between one or two entities. Observations additionally have a timestamp, which is used to represent temporal interactions within the graph. Furthermore, entities may be amended in subsequent batches, by adding a layer of attribute updates.

The efficacy of this model was proven in the last three months since M6, as we accomplished the following quickly:

1) With the help of RiverLoop / TwoSix labs, we added a module that pulls in GitHub pull requests. Building this module took about 10 hours of developer time, and now any OSS project hosted on GitHub can have this information scraped into LAGOON.

2) A “Have I Been Pwned” module was added to collect information on breaches containing the email addresses seen within the OSS community. This module took about 5 hours of developer time, and will continue to work with all emails added to LAGOON in the future.
3) A natural language processing-based toxicity classifier was added to supplement the wordlist-based approach from past results.
4) To showcase how LAGOON might scale out to larger OSS dependency networks than a single project, we ingested the LXML project alongside CPython. This took less than 3 hours of human effort to identify data sources and pull them into LAGOON.

Cross-reference of suspicious Linux kernel contributors

We received a list of 32 suspicious email addresses from Dave Aitel, who investigated the Linux kernel OSS project in collaboration with Margin Research on the SocialCyber program. We checked to see if any of these addresses were reused within the CPython or LXML communities. Unfortunately, none of the addresses from the Linux Kernel Mailing List (LKML) were used in the communities covered by LAGOON’s investigations.

Verifying transfer of domain-specific toxicity from Wikipedia to OSS

In order to test the ability of the BERT-based toxicity classifier to transfer to the target community and genre, all OSS email messages from the CPython community were classified using the BERT tool, and a random balanced set of 100 messages were sampled (50 classified as toxic, 50 not), and independently hand-coded by two researchers on the UVM team, achieving 88% interrater agreement. After reconciling all conflicting categories between human raters, the BERT and exclusion-word toxicity tools were each tested against the resulting human-coded gold standard, resulting in an accuracy score of 69% for the BERT-based classifier and a 40% accuracy score for the exclusion-word-based filter.

Example messages:
1) [NOT TOXIC] “Would Stackless have a better chance of making it into the core if its initial PEP talked only about increasing performance and removing limits on recursion? This advantage is not as "sexy" as continuations, but is also less controversial.”

2) [NOT TOXIC] “First of all, fully specifying all the type checking rules would make for a really long and boring PEP (a much better specification would probably be the mypy source code).”

3) [NOT TOXIC] “I hate to call them 'keyword' arguments when they aren't (check with the\nkeyword module: it will confirm they aren't keywords!-).”

4) [TOXIC] “You Jewish Muslims are your own enemy you get used by Israel as tools and then you get rejected by Muslims too. I honestly don?t know how these people can be so inhuman”

5) [TOXIC] “Those numbers are ridiculous! The only thing they prove is that Congiano should not be programming computers. Anyone getting such results should take a serious look at their algorithm instead of blaming the language. I don't care if it takes 31.5 seconds to compute 36 Fibonacci numbers in Python 2.5.1 with the dumbest possible algorithm”

6) [TOXIC] “Devin, When someone asks me to download a compressed file, its just like the SCAM junk email I get all too often. If the OP would learn how to post on usenet, I would have been happy to help out. I apologize to the group, but the OP needs to learn how to post”

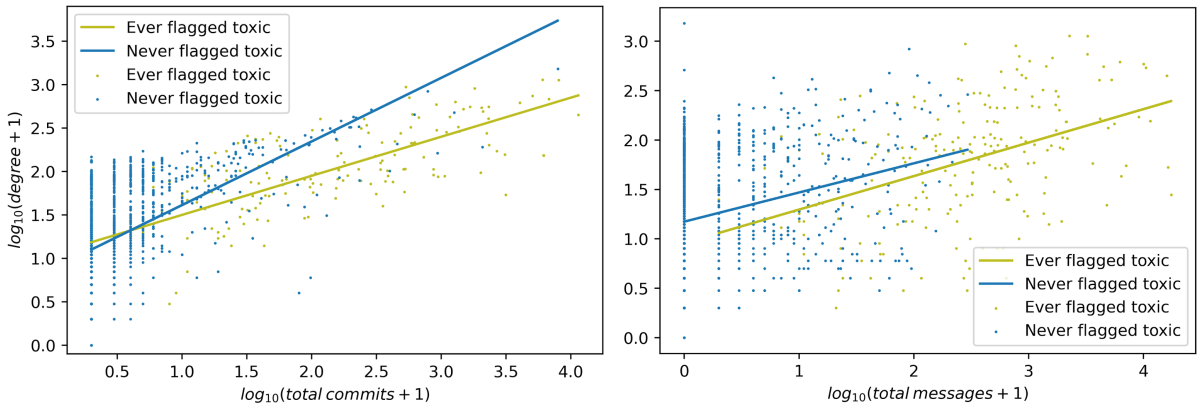
Messages 3, 5, and 6 were misclassified by the exclusion-word filter, but accurately classified by the BERT classifier. Although message 3 contained the word “hate”, it did not actively express antisocial sentiment. Meanwhile, messages 5 and 6 above were classified as non-toxic by the exclusion-word filter, as they contain no explicit words, whereas the BERT-based classifier was able to correctly identify both as toxic in tone.

Updated network analysis using domain-specific toxicity

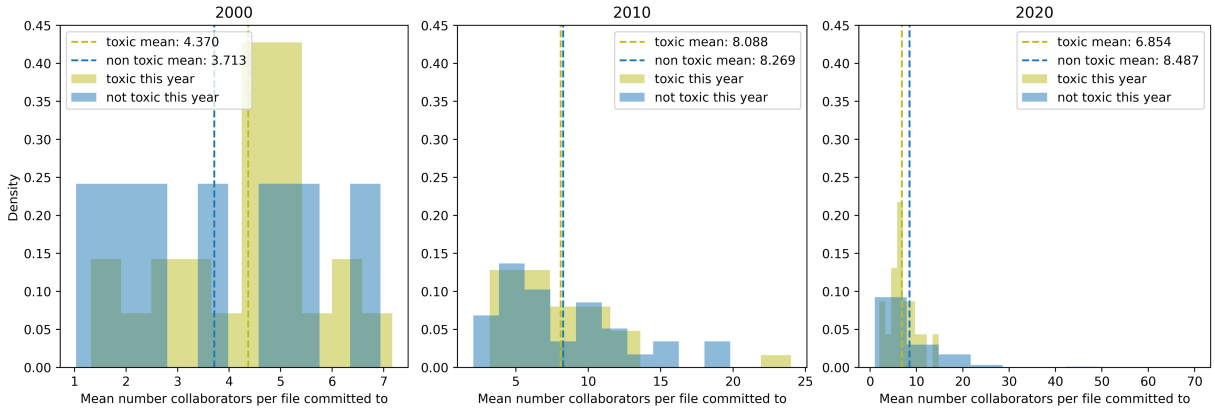
The detail and breadth of the collaboration data available lends itself to several forms of network analysis. At the most fine-grained level, the interactions can be represented as a temporal tripartite network of contributors, commits, and files, where contributors are connected to commits, which are connected to files (and contributors).

To get an overall sense of how users which are flagged toxic differ in their commit-activity from those who are not, we first look at one version of the aggregated unipartite projection network of contributors where two contributors are connected if they have ever committed to the same file in the same year. The ‘degree’ or number of connections in said network represents how many other people they have collaborated with by working on the same file. This value is influenced by both the number of commits that contributors make, but also the number of collaborators per file committed to. In order to characterize this relationship more fully, in the figure on the left below, we look at the degree on the y-axis and the number of commits on the x-axis and find that contributors active on the social layer (the mailing list) are active on GitHub. Looking at the figure on the right below, we see these toxic users tend to work on files with less collaborators.

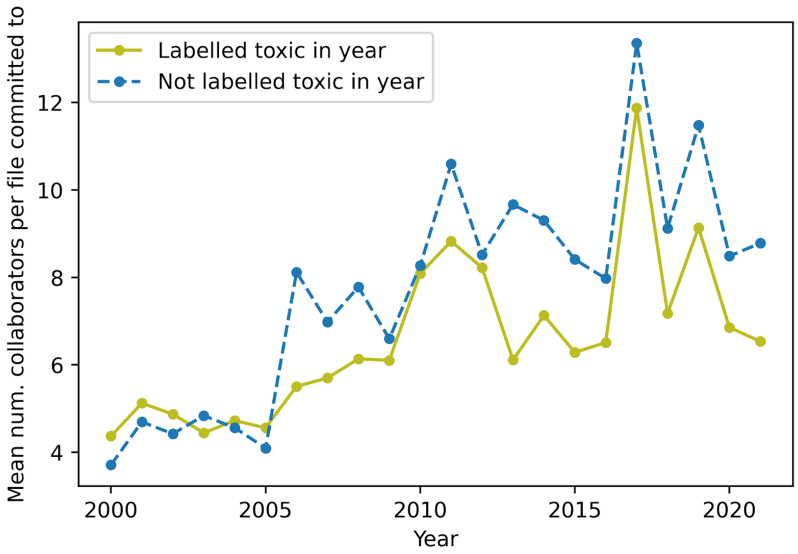
Aggregated over all years



To get a sense of how the number of collaborators per file committed to contributes to this effect, we examined the bipartite projection network of files and contributors, aggregated at the yearly level. We found that past 2005, the mean number of collaborators per file committed to for contributors who had at least one message labeled toxic that year was lower than those who had no messages labeled toxic.

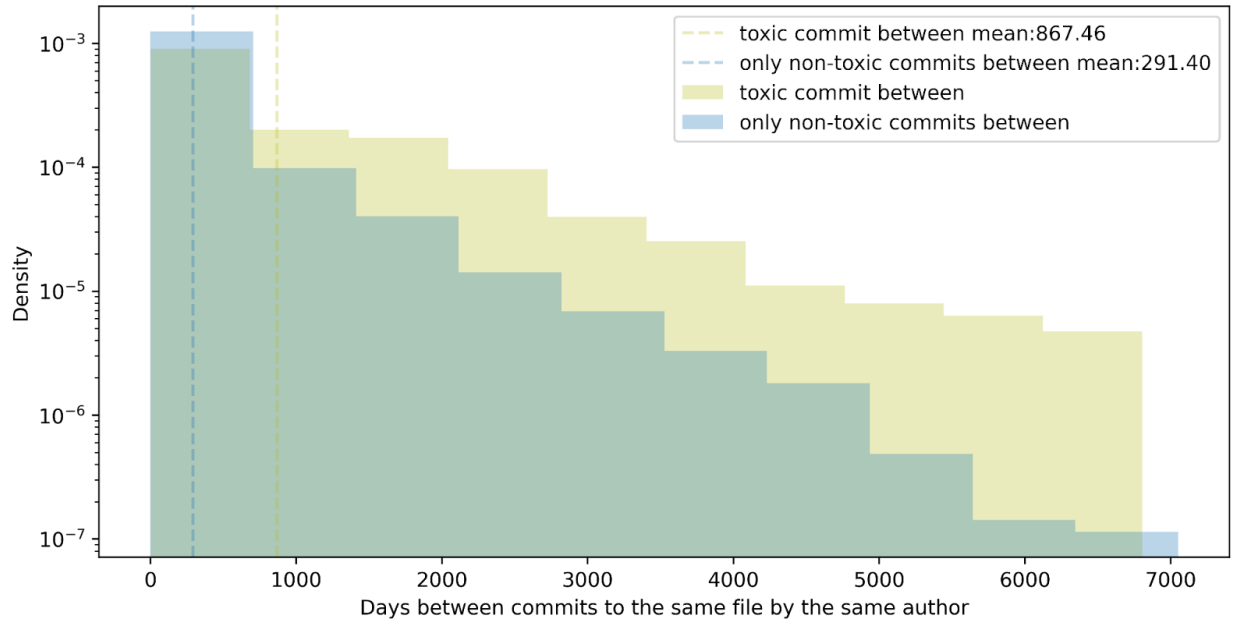


Viewing the mean values of number of collaborators per file committed to for these two groups (those with messages labeled toxic in a given year and those without) over the past two decades we see that this emerged as a trend around 2005 and remains consistent since.



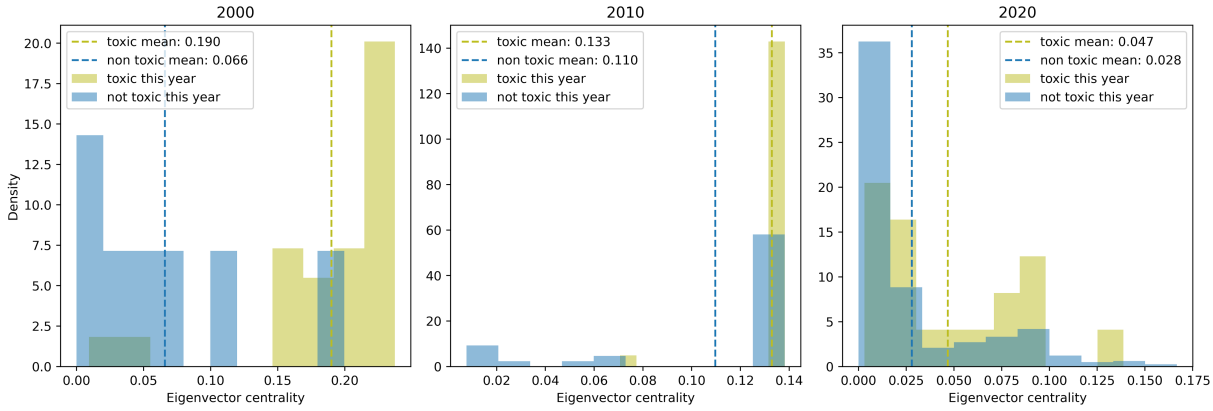
Investigating potential mechanisms explaining why toxic contributors work on files with less collaborators, we find that when a commit is made to a file that includes a commit message labeled as toxic, the time between commits by the same author to that file is higher than the time between commits by the same author for files where non-toxic commits are made. This was determined by examining the tripartite temporal network. Considering every author, file pair where the author committed to that file multiple times. For each of these author, file pairs, we iterated through sequential pairs of commits by that author and checked whether they had a commit from at least one other author between them, ignoring those that did not for better comparison. With the remaining pairs of sequential commits, we recorded the time difference and categorized them by whether or not there was a toxic commit between them. This analysis

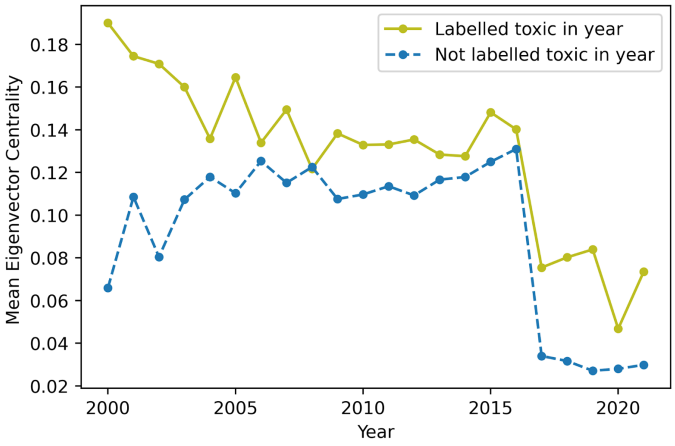
highlights one key effect of toxicity: slowing down (and potentially ending) collaborations.



Considering that the contributors engaging in toxic behavior had higher levels of activity and more connections, but lower number of collaborators per file, one consequence of that from a network perspective is higher centralization according to other network centrality measures besides degree. Looking again at the unipartite projection co-commiting network, now aggregated at the yearly level, we calculate the eigenvector centrality of contributors for each year and group them by whether or not they sent messages labeled toxic that year.

Eigenvector centrality of node v in the network represented by adjacency matrix A is the v^{th} value in the vector **x** which satisfies the following matrix equation: **Ax** = λ**x**, and represents centrality both in terms of number of connections but also centrality of those connections. This vector of centralities was calculated and histograms were generated for each year, three of which are shown below. Additionally, we plotted the mean values for these categories for the past two decades and found that nodes with messages labeled toxic are consistently more central than nodes without, though the network as a whole becomes less centralized in 2017 when the number of contributors increased dramatically. One mechanism for this increase in centrality for toxic nodes is again related to the slower productivity and overall disengagement of their collaborators. Toxic nodes can gain in centrality simply because others slow down or leave.





Analyzing Have I Been Pwned data

The Have I Been Pwned (HIBP) database, accessible via the website https://haveibeenpwned.com/ or its associated APIs, can be used to obtain the ‘breaches’ and ‘pastes’ that any email address has been involved in. Breaches are incidents where data is inadvertently exposed in a vulnerable system, while pastes constitute information that has been pasted to a publicly facing website designed to share content such as Pastebin. Such personal information getting exposed may raise the overall vulnerability of the owner of the email address, which may increase their chances of being a recipient of toxicity. This led us to investigate the correlation between toxicity and the number of breaches a person’s email has been involved in.

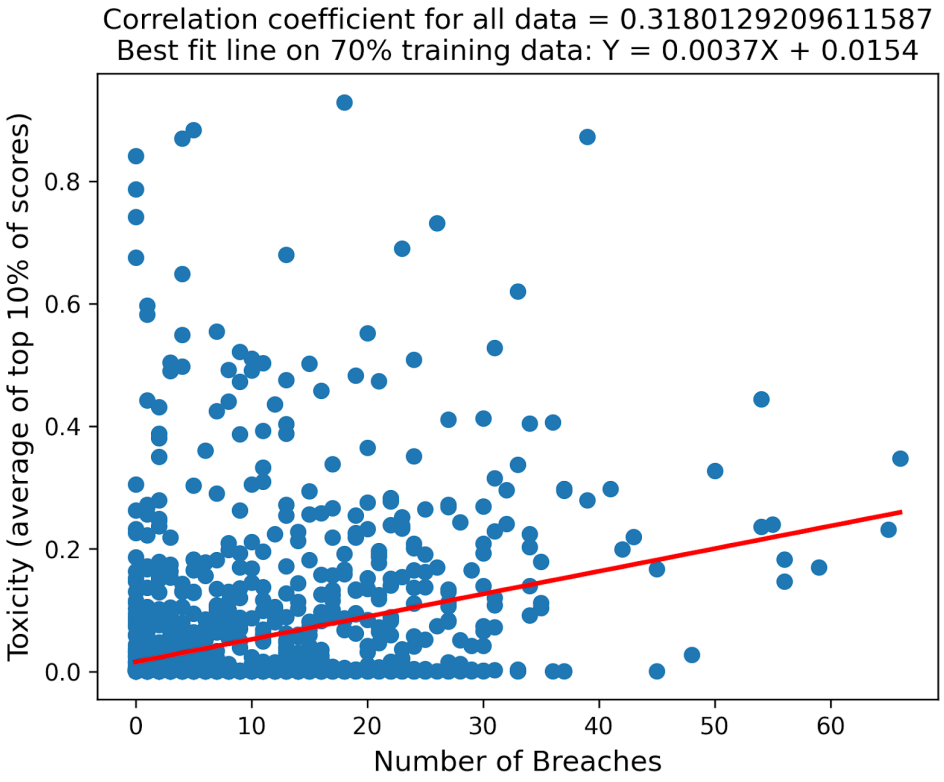
Toxicity information is taken from the text entities – mailing list and commit messages – in the 1-hop neighborhood of a person in LAGOON’s sociotechnical graph. We used a single feature – the score of the aforementioned BERT-based toxicity classifier, which is between 0 and 1 with a higher score indicating a more toxic message. Toxicity scores were aggregated across all

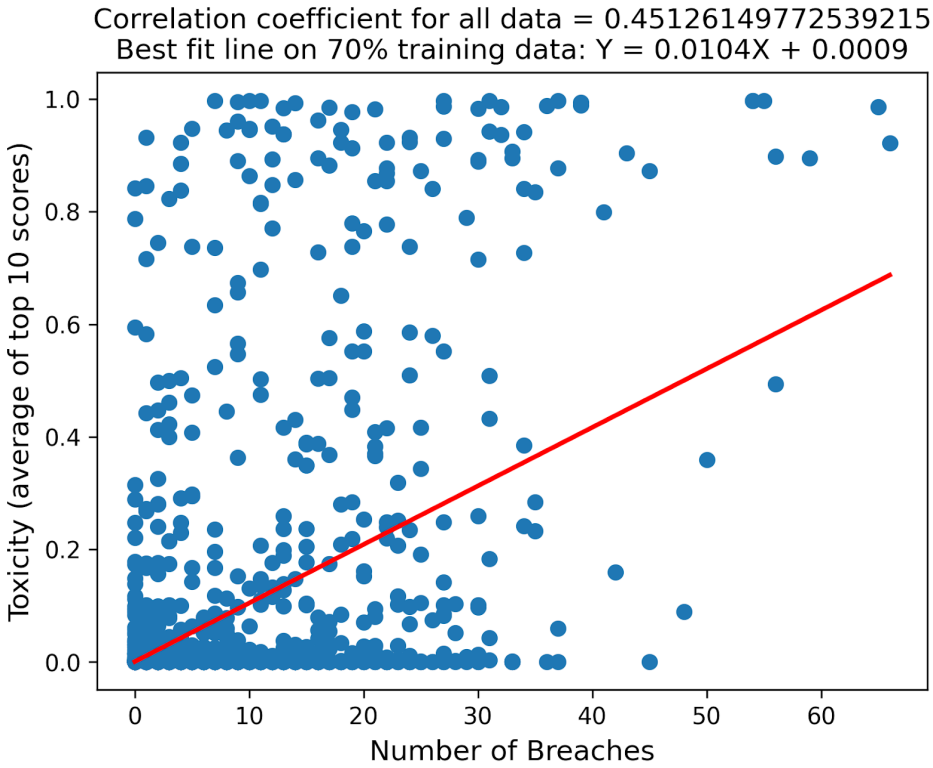
messages in the neighborhood by taking the average of either a) the top 10% of scores, or b) the top 10 scores.

For obtaining breaches, we restricted the emails to only those for the 1772 persons with at least one commit in the entire temporal history of the sociotechnical graph. This helps in weeding out fake persons with spam email IDs. Our improved entity fusion module also helped in weeding out spam.

The plots below show results of regressing aggregated toxicity on the number of breaches. Note that we used the number of breaches as the independent variable. The scatter plots show all the 1772 persons. The red lines are the best fit lines obtained from training a linear regressor on 70% of the data; the remaining 30% is used as test data.

The top figure aggregates the top 10% of toxicity scores, which has 31.8% correlation with the number of breaches. The coefficient of determination (R^2) is 0.1 for both training and test data. The bottom figure aggregates the top 10 toxicity scores and increases these numbers to 45.1% correlation and R^2 value of 0.2 for both training and test data.





The noteworthy result from the above plots is that getting one's email breached, which is a proxy for having a longer-term internet presence, is tied to receiving higher levels of toxicity. This in turn may lead to vulnerabilities / instabilities in an OSS community.

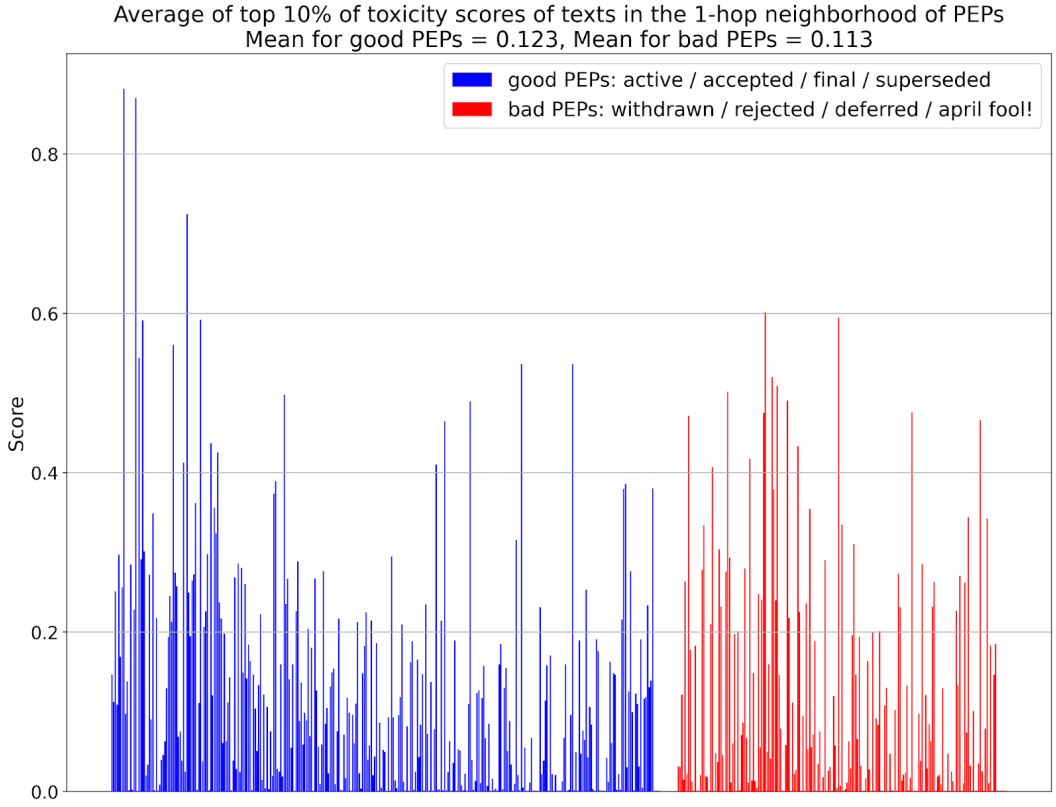
The bottom figure is included to illustrate an alternative view, which does not look at the distribution of toxicity but rather the soft maximum of toxicity. Since the soft maximum is affected by the volume of messages a user is exposed to, and the volume of messages is also roughly correlated to that user's general level of activity, it becomes easier to show that breaches are related to toxicity, because the underlying hypothesis is that activity implies activity. Unfortunately, this hinders our ability to assess the effects of one-off toxicity incidents: because users who are more active are more likely to have experienced toxicity, it's difficult to find a very active user who has not experienced toxicity, which would be necessary to begin discussing the effects of one-off toxicity.

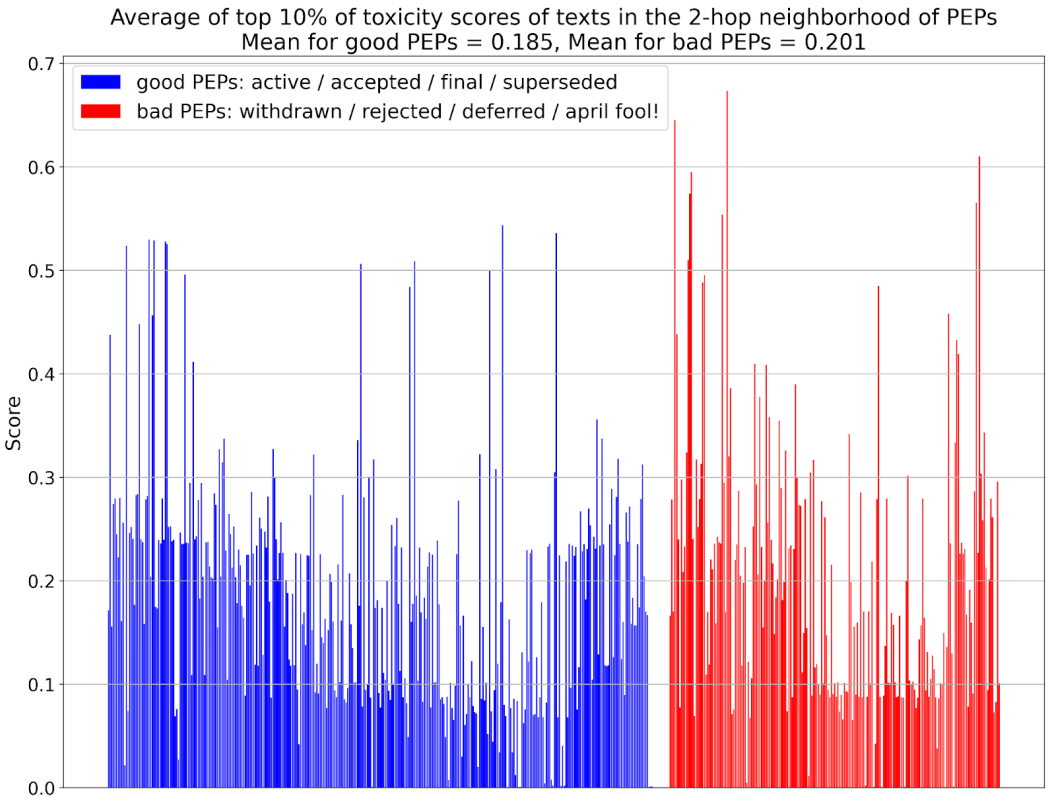
Re-analyzing PEPs and contributor disengagement using new toxicity features

We had previously used toxicity features from bad word lists to differentiate 'good' PEPs from 'bad' PEPs – these results are described in more detail in the latter section on Past Results. Briefly, PEPs are Python Enhancement Proposals, we classify them as good if their status is 'Active', 'Accepted', 'Final' or 'Superseded', and bad if their status is 'Withdrawn', 'Rejected' or 'Deferred'.

We re-ran the earlier analyses after replacing the bad word list features with toxicity scores obtained from the BERT-based classifier. Scores are considered in both the 1-hop and 2-hop neighborhood of PEPs in the sociotechnical graph, i.e. messages referencing a PEP, and messages referencing such messages (as before, the term ‘messages’ here includes both mailing list and commit messages present inside LAGOON). Toxicity scores were aggregated by taking the average of the top 10% of scores.

The results are shown below for the 1-hop and 2-hop neighborhoods in the top and bottom figure, respectively. While the 1-hop results do not indicate that bad PEPs have more toxicity surrounding them (in fact, they weakly indicate the contrary), the 2-hop results do show that bad PEPs have more toxic messages around them.





The nature of these plots is interesting, particularly because our past results described later indicate that bad word based toxicity does not differentiate between PEPs. In general, such toxicity aggregation methods perform inferior to ML based techniques since, as will be shown later, the former do not consider the constellation of toxicity around a user, while the latter do. However, BERT based toxicity classifier scores perform better than bad word based toxicity, which we hypothesize is because the former does not suffer from the limitations of the latter which can indicate a message such as “It’s not possible to destroy static types at Python exit” as toxic due to the presence of the word ‘destroy’.

We also attempted to use the BERT based toxicity classifier scores to predict contributor disengagement. While our past results had used bad word counts as features for several ML techniques, the current results perform a simpler linear regression of disengagement on toxicity. The regression models were not very successful.

Investigation of potentially-overlooked source code changes

One potential vector of attacking an OSS project is to find a file which has been abandoned by core maintainers. Since files that have been abandoned for a long time might have less stringent quality checks, pushing malicious code to these code regions might be easier.

LAGOON is now packaged with a report that looks at all tracked source code files, and compiles a list of commits and maintainers that have touched those files within the last 3 years. Then, starting with those files who have the fewest unique maintainers, the script prints statistics about

contributors to those files who have only touched the file once throughout the project's history. The result looks like this:

Sorted by (files with fewest unique maintainers, maintainers with fewest commits)

Rows of (User ID, number of user commits, file they touched one time, unique contributors to file, commits against file)

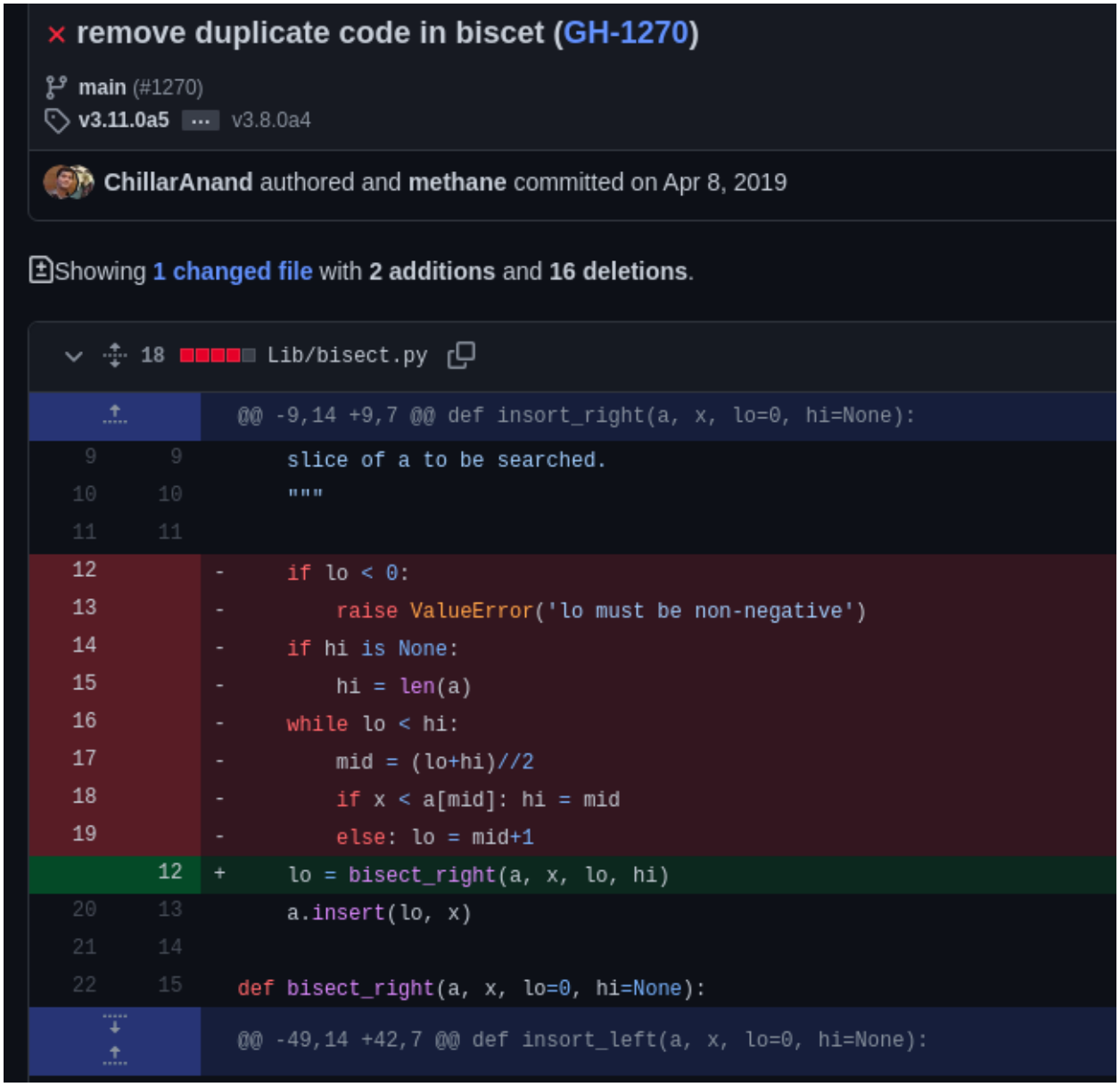
```
[
  0 : "(132743, 1, 'Lib/colorsys.py', 2, 2)"
  1 : "(131420, 1, 'Parser/pgen/pgen.py', 2, 6)"
  2 : "(17466367, 1, 'src/lxml/tests/fuzz_xml_parse.py', 2, 2)"
  3 : "(131538, 1, 'Python/pystrtod.c', 2, 5)"
  4 : "(131097, 1, 'Parser/pgen/automata.py', 2, 5)"
  5 : "(132550, 1, 'Lib/colorsys.py', 2, 2)"
  6 : "(132852, 1, 'Lib/bisect.py', 3, 4)"
  7 : "(17466353, 1, 'setupinfo.py', 3, 7)"
  8 : "(131606, 1, 'Lib/bisect.py', 3, 4)"
  9 : "(132611, 1, 'Tools/unittestgui/unittestgui.py', 3, 3)"
  10 : "(131433, 1, 'Lib/genericpath.py', 3, 5)"
  11 : "(131584, 1, 'Tools/demo/rpython.py', 3, 3)"
  12 : "(17466371, 1, 'src/lxml/html/__init__.py', 4, 6)"
  13 : "(132823, 1, 'Lib/asyncio/transports.py', 4, 6)"
  14 : "(17466501, 1, 'src/lxml/tests/test_etree.py', 5, 29)"
  15 : "(131075, 1, 'Lib/test/test_importlib/test_zip.py', 5, 14)"
  16 : "(132455, 1, 'Lib/html/parser.py', 5, 8)"
  17 : "(131266, 1, 'Lib/socketserver.py', 5, 9)"
  18 : "(131432, 1, 'Lib/http/__init__.py', 6, 11)"
```

These files may then be investigated in the UI to find potentially malicious commits to areas of the code which are in regions that may be overlooked by the community. For example, `Lib/bisect.py` is a very highly used library that does not receive many updates. Yet, user 132852, Sergey Golitsynskiy, pushed exactly one commit to CPython that affected this core module. Through the UI, we can see (and verify) that this was a documentation change:

<lagoon.db.schema_fused.FusedEntity 120499: EntityTypeEnum.git_commit 6a1e9c2673>

time	1594509511
message	Fix error in docstrings in bisect module (GH-21422) The docstrings for `bisect_right()` and `bisect_left()` contain sample code that has a bug: `a.insert(x)` should be `a.insert(i, x)`.
commit_sha	6a1e9c26736259413b060b01d1cb52fcf82c0385

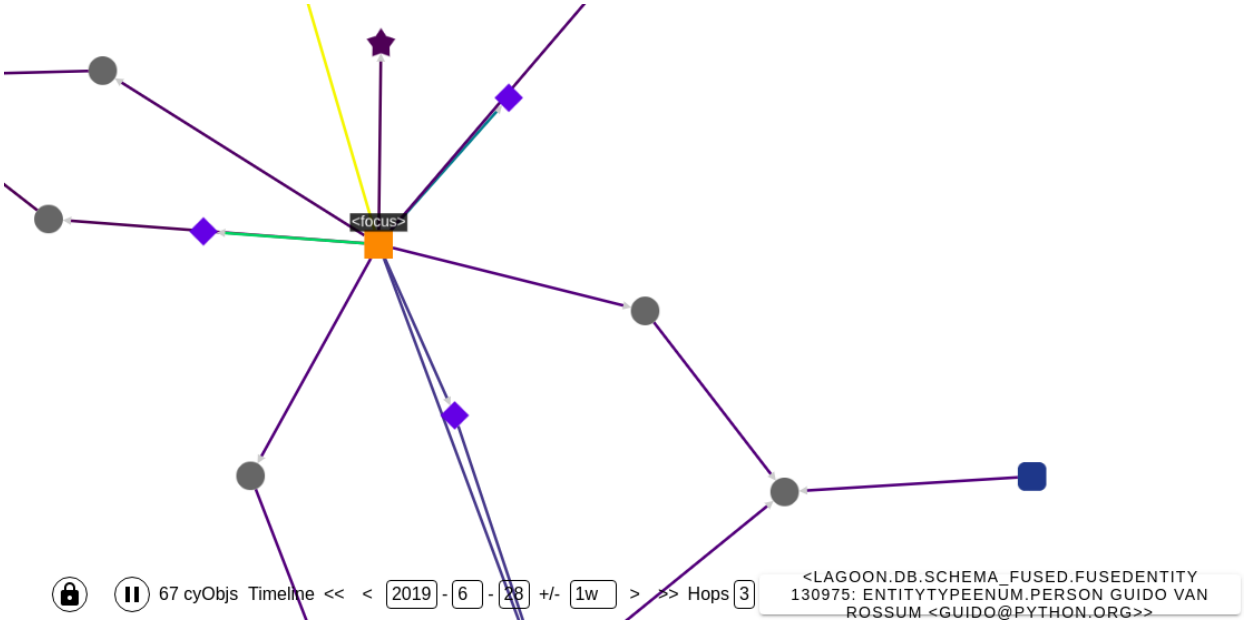
However, there was another change from a new maintainer to `Lib/bisect.py`, by user 131606. Using the new GitHub integration, we can see that this commit removed duplicate code:



While generally these changes look safe, this still demonstrates the ability of a SocialCyber-derived tool such as LAGOON to provide statistical priors that point to commits requiring greater scrutiny due to the sociotechnical histories of their contributors.

Continued improvements to user interface

The LAGOON interface has received multiple updates since the M6 milestone, including:



- Data is now more accessible via a default focus lock on the main element. Since focus is locked by default, the default action is to show more information on a selected node. As the sociotechnical graph used by LAGOON is very dense, we found this to be the more common mode of user interaction.
- Batch information is now annotated by all fusions in the UI. This makes finding the provenance of different data snippets significantly easier.
- Entity search is now significantly faster, due to the addition of a full text index with greater flexibility than in previous versions.
- Entity search now looks at multiple aliases for the same fused entity. For example, “Stefan Behnel” often goes by the alias “scoder”; both of these terms will now find that person.
- Commit objects now have a clickable link to the commit on GitHub, which provides access to additional tools for source code repositories.

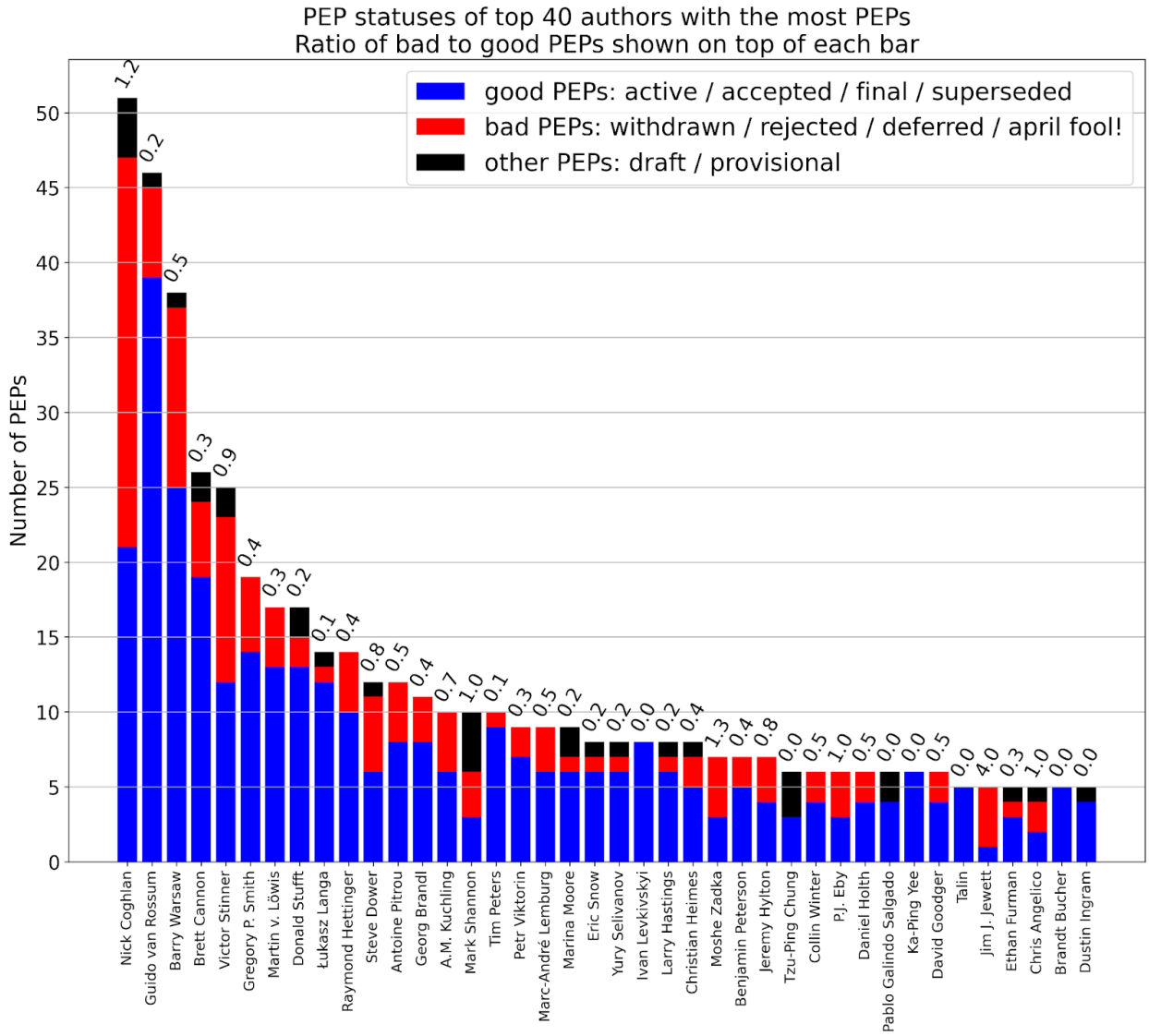
Past Results

Included below are sections from the M1, M3, and M6 reports, near verbatim, which are relevant to the high-level objectives and accomplishments of the LAGOON program. That is, these sections allow for interpretation of this document as a standalone report with all findings.

Organizational structures amongst Python collaborations

Our efforts toward uncovering organizational subgroups proved successful at discovering collaborative relationships amongst contributors. We focused particularly on how these relationships develop over time. These relations were considered with specific interest in identifying potentially subversive elements, though we have no conclusions supporting the existence of subversive elements within the CPython community at this time.

Since the growth of Python is predominantly organized around PEPs, we first looked at which community members were proposing alterations to the language, and the success rate of their proposals. In a healthy community, initial rejections can be constructive, leading to positive contributions in the future. On the other hand, unhealthy communities might transition from rejections to disengagement with the project. Conversely, regardless of community health, a contributor which pushes for many PEPs that are rejected may be trying to fatigue the community into eventually accepting a compromising idea.

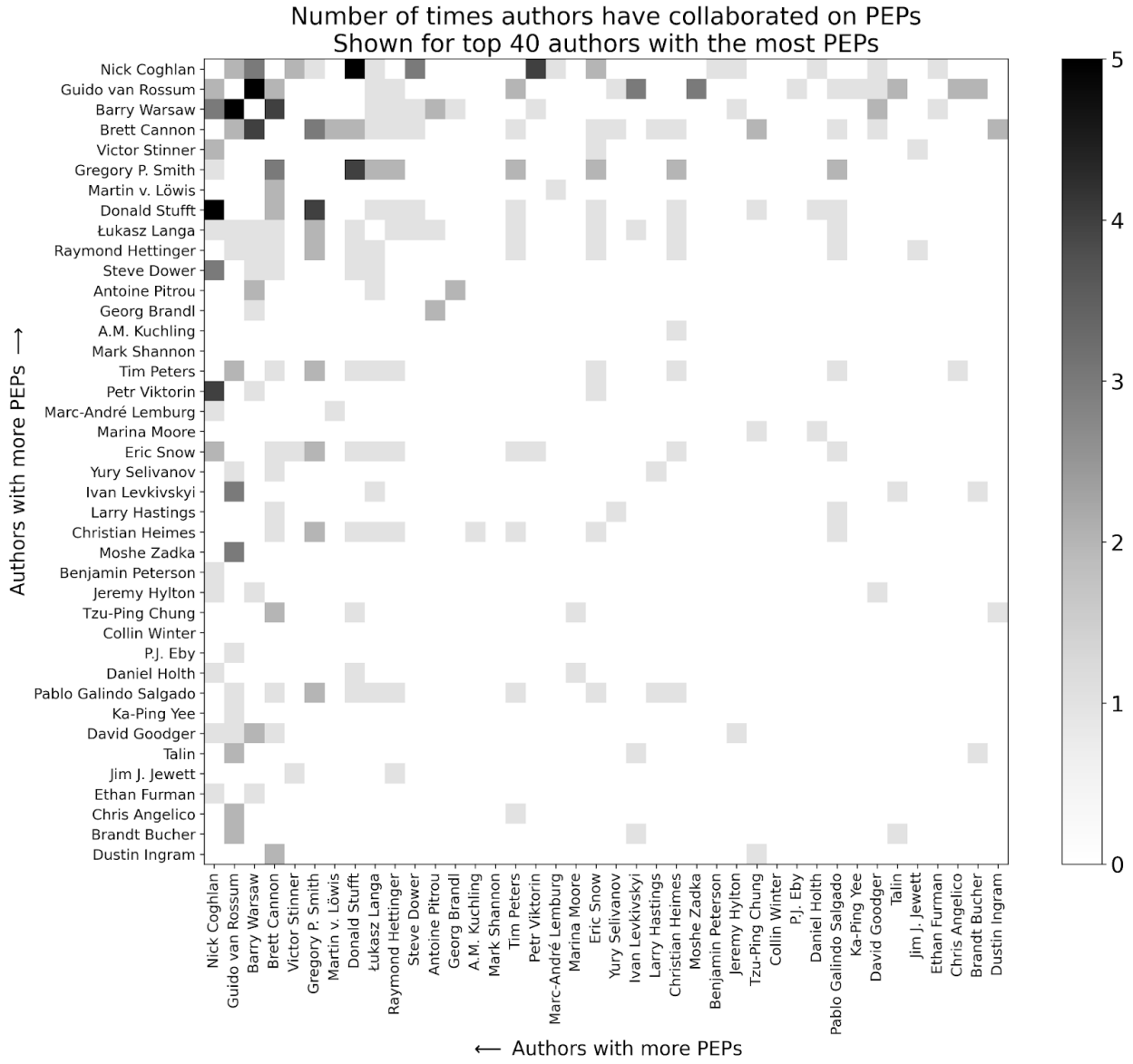


An interesting dichotomy appears here, where the majority of PEP proposers are usually successful with their bids to affect the community. This could be a result of the barriers to creating a PEP -- one probably wants to secure sufficient buy-in on an idea before going through the formal process. The result is reminiscent of survivorship bias, in that the majority of proposed changes to Python which can be tracked are already in the late stages of ideation. This phenomenon might be unavoidable, as mailing lists and larger discourse necessarily

happen after proposed design changes become public, rather than during the creation of the design.

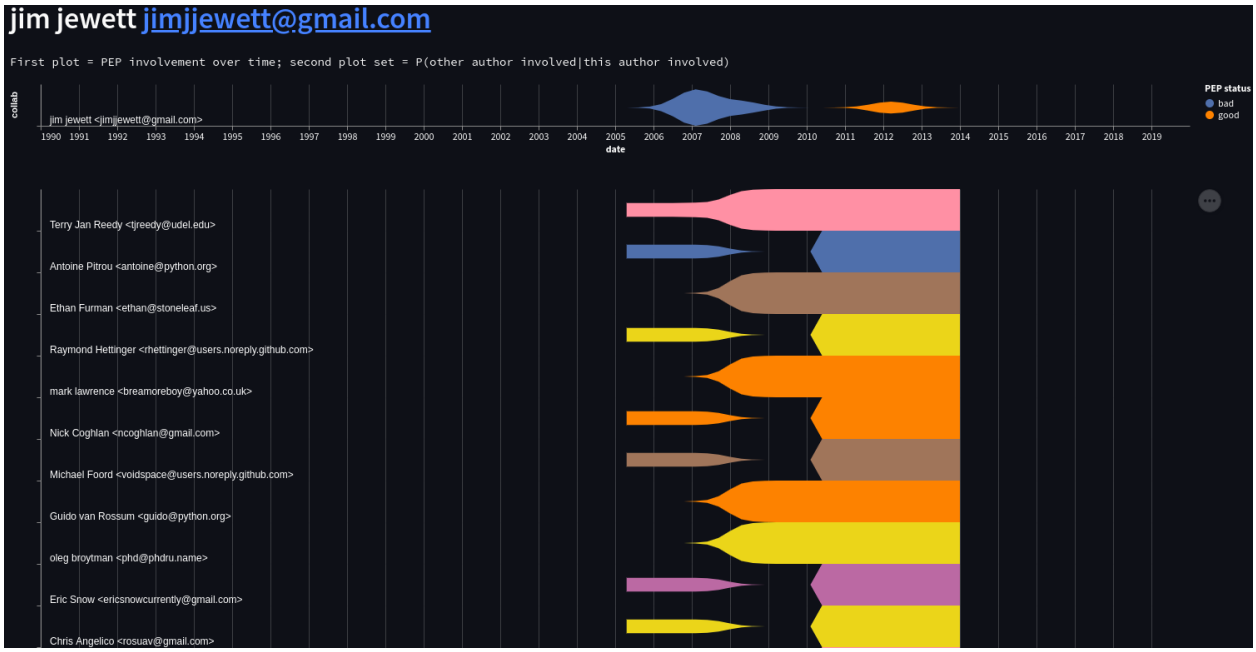
Looking at accept/reject rates does make apparent a few potentially suspicious contributors: Nick Coghlan, Moshe Zadka, and Jim Jewett have a greater than 50% reject rate. This could be part of the learning process, but it's interesting that so few core Python contributors follow this pattern.

Most PEPs are written by a single author, but some have multiple. If we look at the following symmetric matrix of collaborations amongst the top 40 authors, with the diagonal removed, we can see that multi-author PEPs do not give a strong indication about groups of authors which frequently collaborate. Note that the axis only goes to 5 here, whereas many PEP authors have written significantly more than 5.



PEPs with multiple authors had somewhat different accept/reject statistics. Notably, while the general accept:reject ratio for multi-author PEPs was around 2.4:1, those co-authored by Python's BDFL, Guido van Rossum, had an accept:reject ratio of 29:1.

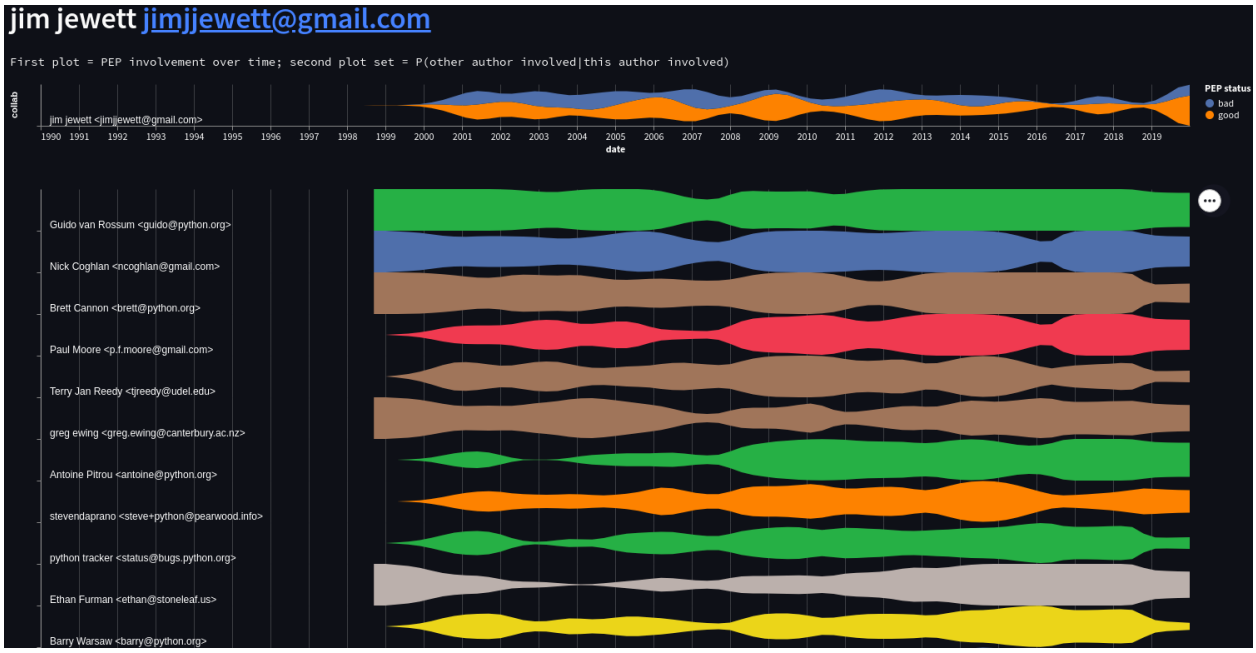
In addition to direct collaborations through authorship, all PEPs have significant discussions on the various Python mailing lists, and in particular on python-dev@python.org and python-ideas@python.org. These two mailing lists are part of the OCEAN data from UVM, so we also scraped their contents to find which community members were involved in the discussions of all PEPs. From there, we can look at interactions amongst community members over time as it relates to specific enhancement proposals. We tried plotting this as a graph structure, but most community members were too connected to make sense of the resulting visual. Instead, we looked at characterizing the organizational group around specific contributors. For example, here is an analysis of PEPs created by Jim Jewett, one of the above PEP authors who submitted many PEPs that were ultimately rejected:



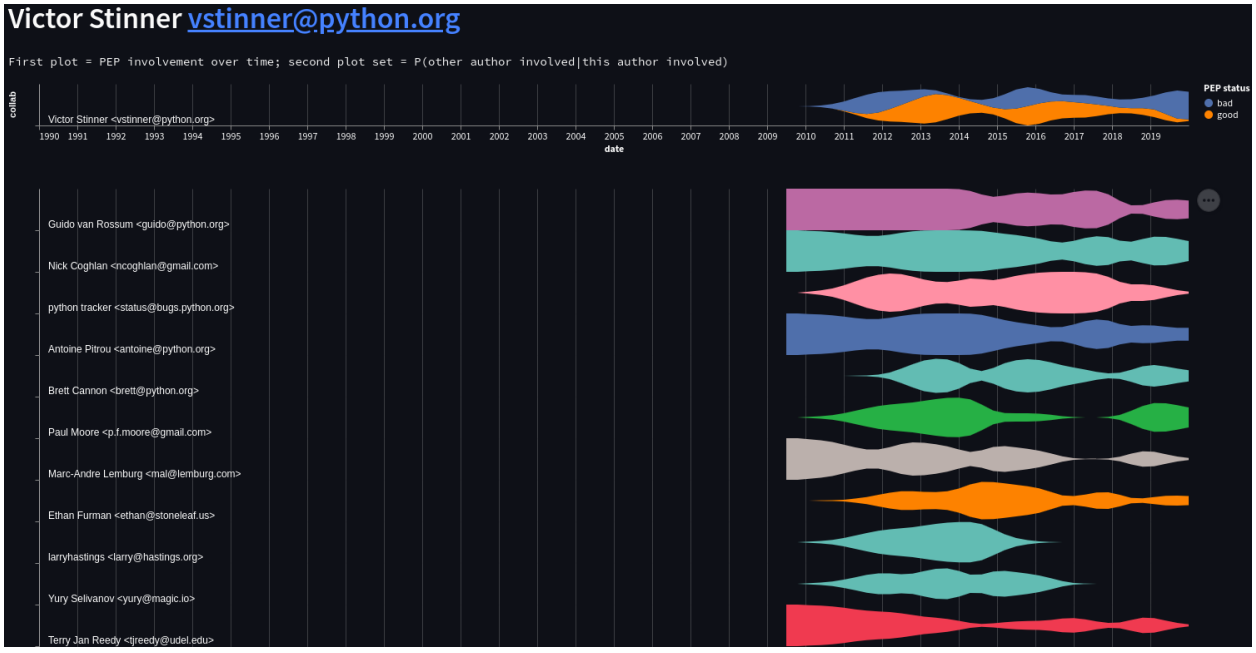
The top plot shows how many PEPs Jim submitted over the years as a density plot with a Gaussian kernel of half a year being one standard deviation, and is colored according to whether or not their current statuses reveal them as accepted or rejected by the community. The lower density plots show the probability that different community members were involved in these same PEP discussions, and are sorted such that the members who most commented when this member commented are at the top; i.e., descending *P(other contributor wrote something about a PEP|this contributor wrote something about that PEP).*

We can see that Jim submitted 4 PEPs between 2007 and 2008, all of which were rejected. These PEPs had less of the community involved in discussions around them. In contrast, Jim's accepted PEP from 2012 had discussions with participation from more of the community. To draw further conclusions about Jim Jewett's role within the Python community, we can look at a

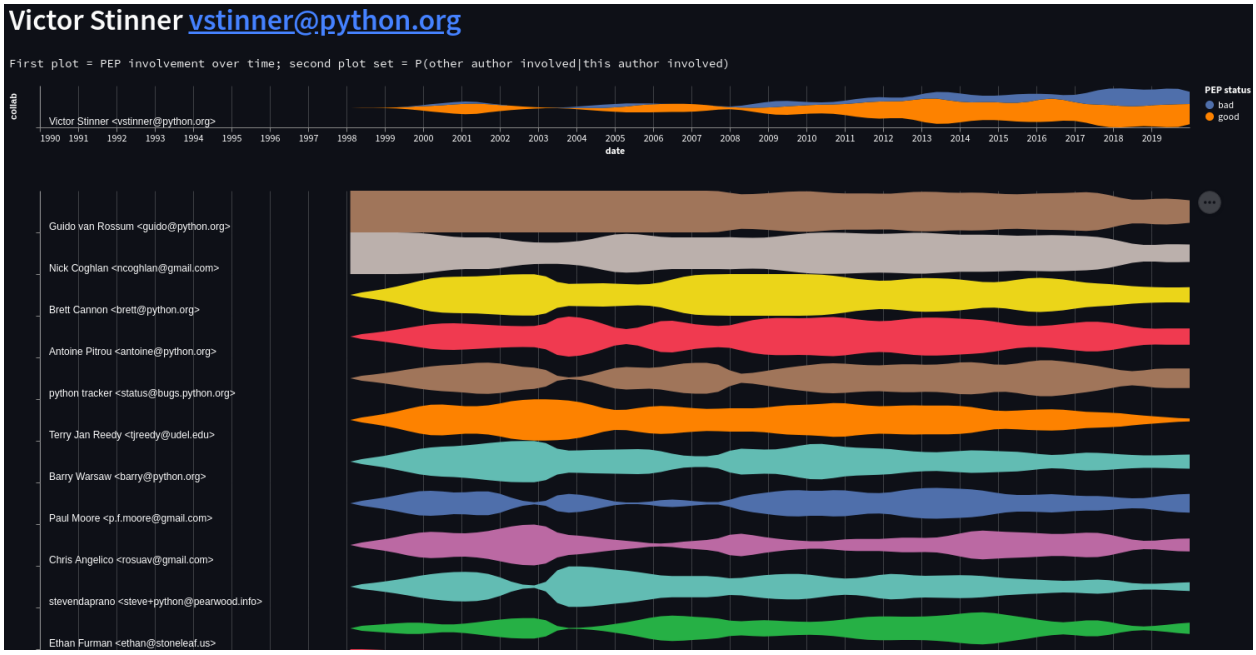
similar plot plotted against all PEPs in which Jim participated in discussions, rather than only PEPs created by Jim:



This demonstrates that Jim has significant involvement with the Python community outside of his own PEPs. Interestingly, the PEPs that Jim gets involved with tend to involve much of the community. This might be a symptom of Jim not being a committer to the CPython repository, but rather a contributor limited to the exchange of ideas with the community. For contrast, here's a report for PEPs proposed by Victor Stinner, a core contributor with over five thousand commits in the CPython repository:



And a matching report for Victor's involvement in all PEPs, whether or not they created them:



From these, we can start to tell a story: Victor was tangentially involved with the community since 1999, and began significant contributions to Python between 2008 and 2009. Victor's first commit to CPython was from 2010, a year before his first PEP. Unlike Jim's PEPs, which fostered discussion amongst many community members, conversation around Victor's PEPs involved only a few key contributors. As time passed and Victor became more integrated with the project, the number of community members involved in discussion of their PEPs declined, though their PEPs were still often accepted. This illustrates that Victor is a fundamentally different kind of contributor than Jim. Jim tested proposing major changes to Python a few times, but only saw a 20% success rate across 5 PEPs. Jim continued to engage with the community afterward, but stopped submitting PEPs. Victor's first PEP was also ultimately rejected, but they continued submitting PEPs, averaging out to just over a 50% success rate across 25 PEPs. Whereas Jim's involvement in PEPs was primarily restricted to large discussions, Victor was involved in a larger number of discussions involving fewer contributors. One way to divide these two patterns would be to consider Jim a "trending topic" contributor, who engages primarily in topics which affect and are of interest to many users, whereas Victor is more of a "broad topic" contributor who engages in a much more diverse set of topics, many of which have fewer users interested in them.

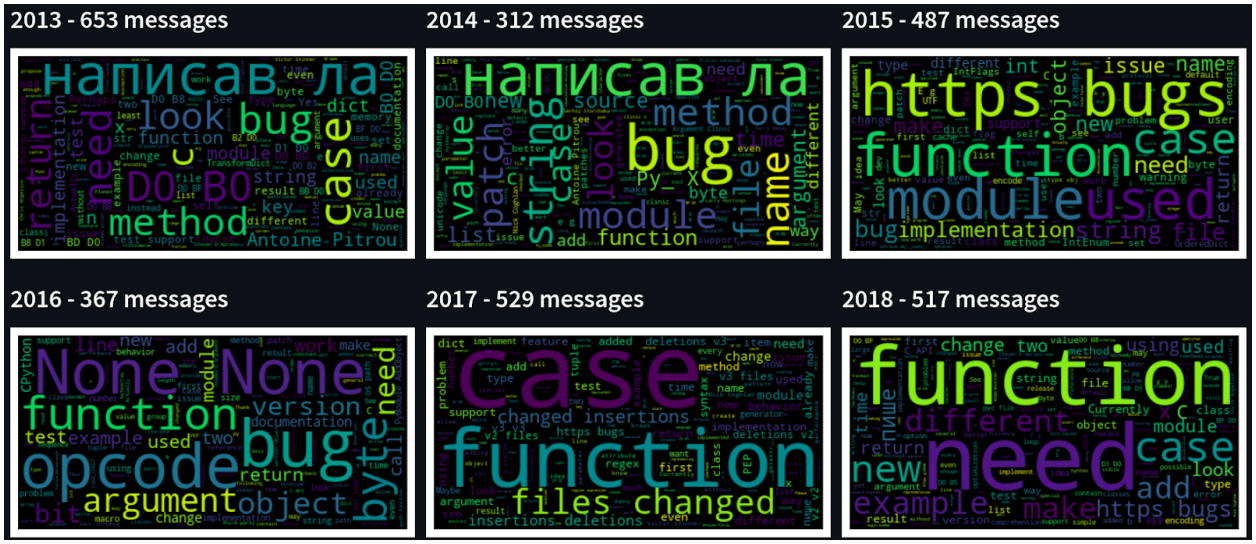
If we flip back to Jim, and poke around the mailing list during one of their PEP discussions, most of Jim's messages and responses were technically-centered and bland from a toxicity / charged emotions point of view. This could imply that Jim is a competent engineer without the language design chops to advocate effectively for their PEPs.

For an example of a contributor with more interesting mailing list content, we consider Serhiy Storchaka, who only helped author a single PEP, but whose community involvement otherwise looks like Jim's. It wasn't difficult to find Serhiy's technical knowledge being called into question with some mild toxicity (the word "suck"):

> On 2/11/2016 11:01 AM, Ethan Furman wrote:
> > On 02/11/2016 10:50 AM, Serhiy Storchaka wrote:
> > > I have strong preference for more strict and simpler rule, used by
> > > most other languages -- "only between two digits". Main arguments:
> >
> > > 2. Most languages use this rule. It is better to follow non-formal
> > > standard that invent the rule that differs from rules in every other
> > > language. This will help programmers that use multiple languages.
> >
> > If Python followed other languages in everything:
> >
> > 1) Python would not need to exist; and
> > 2) Python would suck ;)
> >
> > If our rule is more permissive that other languages then
> > cross-language developers can still use the same style in both
> > languages, without penalizing those who want to use the extra freedom
> > in Python.

This doesn't make Serhiy subversive or indicate that this is the reason Serhiy stopped submitting PEPs though; it only supports that they are perhaps a more junior language developer, whose strong opinions evoked lackluster responses that stymied further PEP submissions.

We briefly explored using word cloud techniques to attempt to capture Serhiy or any other author's changing interests over several years; we include this to illustrate the point that such a technique might be used to approximate concerns at the top of a contributor's mind, however, more filtering is probably needed to pull significant utility out of such an approach:



A better approach might be to use e.g. Shifterator, which filters out common words based on a reference corpus, and might let us better differentiate the technical interests of different contributors.

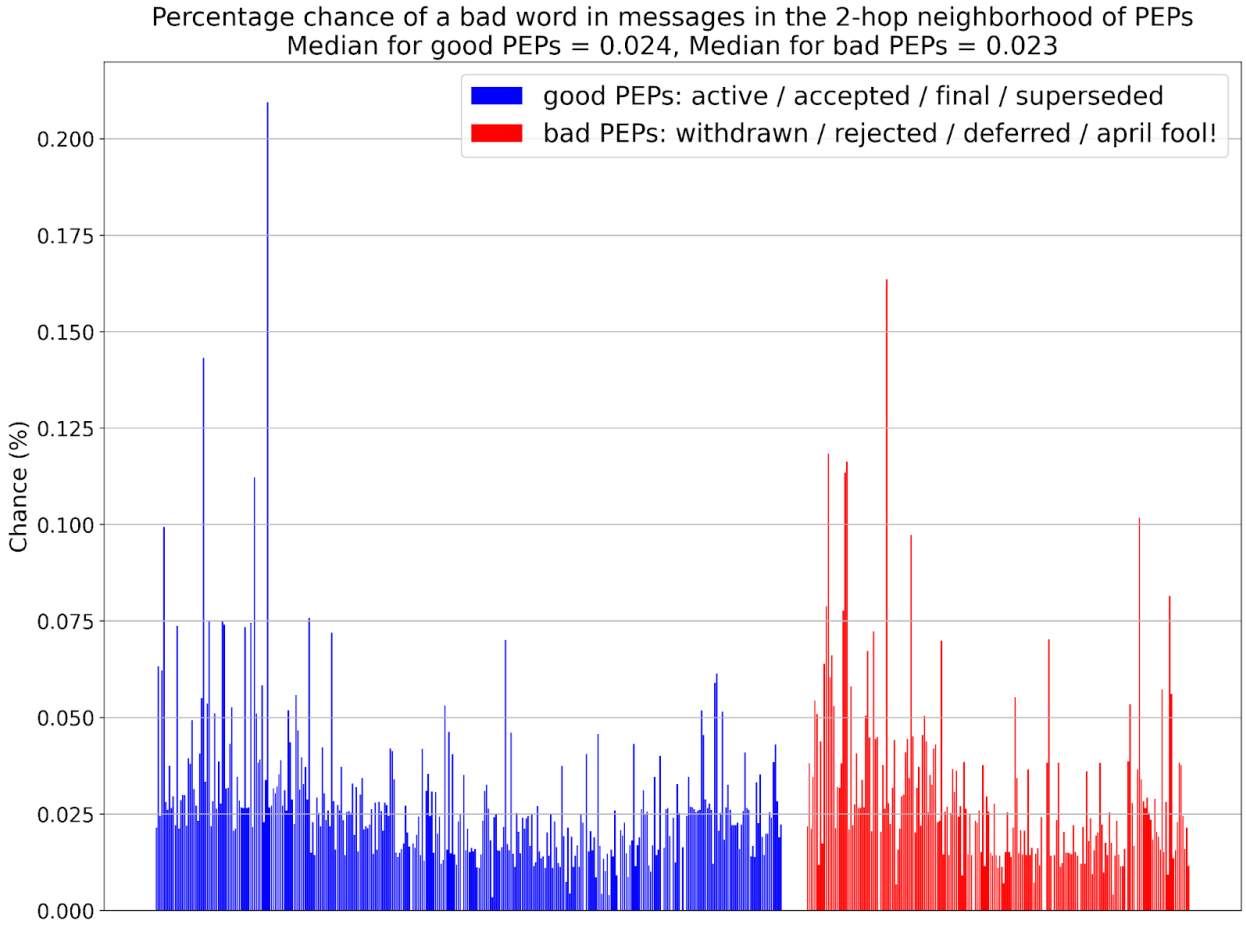
To summarize our thoughts on the organizational structure of Python:

- Organizational structure in Python seems to be predominantly informal rather than formal, due to the large number of messages discussing PEPs and the small list of people actually writing PEPs.
    - It seems that architects of specialized aspects of Python, like ctypes' Thomas Heller, are discussed according to the same structure.
- The majority of PEPs are accepted, though a healthy number are rejected. This demonstrates that Python's PEP approval system is discriminative, rather than a mere formality.
- Due to the informal organization of PEP discussions, attempting to find cohesive, disparate "groups" of contributors is very difficult, and such groups certainly would not have clean separations between them.
- The influence of one contributor on another can be reasonably measured by looking at how often their interactions coincide; this yields a distance metric which can be used to conjecture about the relative influence between two contributors in the community.
    - This is borrowing from topology discussions with Michael Robinson a bit; it's interesting that we can extend topology into the temporal domain relatively simply by leveraging kernel densities.

Aggregate Toxicity versus Constellations of Toxicity

Briefly, we want to re-emphasize a point with respect to the value of machine learning within SocialCyber as it relates not only to generating features / sentiment analysis in messages, but also to predicting contributor or organizational outcomes. It would be great if there were some "simple" formula which demonstrated that specific constellations of interaction, or that toxicity in general amongst a community were indicative of the community's sentiment toward that topic.

However, we found that **aggregate toxicity in messages discussing a PEP could not be used to infer its success**. For example, below is a plot showing toxicity in messages regarding PEPs that were accepted vs those that were rejected. Both the 1-hop and 2-hop plots (meaning immediate discussion relating to a PEP and messages discussing those messages) look similar to the following:



There's essentially no divide between the emotional strength of the community's discourse around a PEP and whether or not the PEP is accepted. We found a similar situation with respect to toxicity and disengagement -- that is, the aggregate toxicity around a user was not useful to predict that user's disengagement. **However, ML methods which considered the full constellation of toxicity around a user were able to account for 11% of the variation** in disengagement, and we would expect something similar for predicting PEP accept/reject rates. This is **due to ML's ability to exploit constellations of toxicity in the sociotechnical graph, instead of being limited to simple aggregates**. Even more of the variation could probably be explained with a better definition of toxicity: here, toxicity is computed in terms of counts of specific bad words, which may have actually been used in an innocuous context. For example, one of the tall bars in the figure is due to specific words being marked as toxic, though they were used in non-toxic technical contexts:

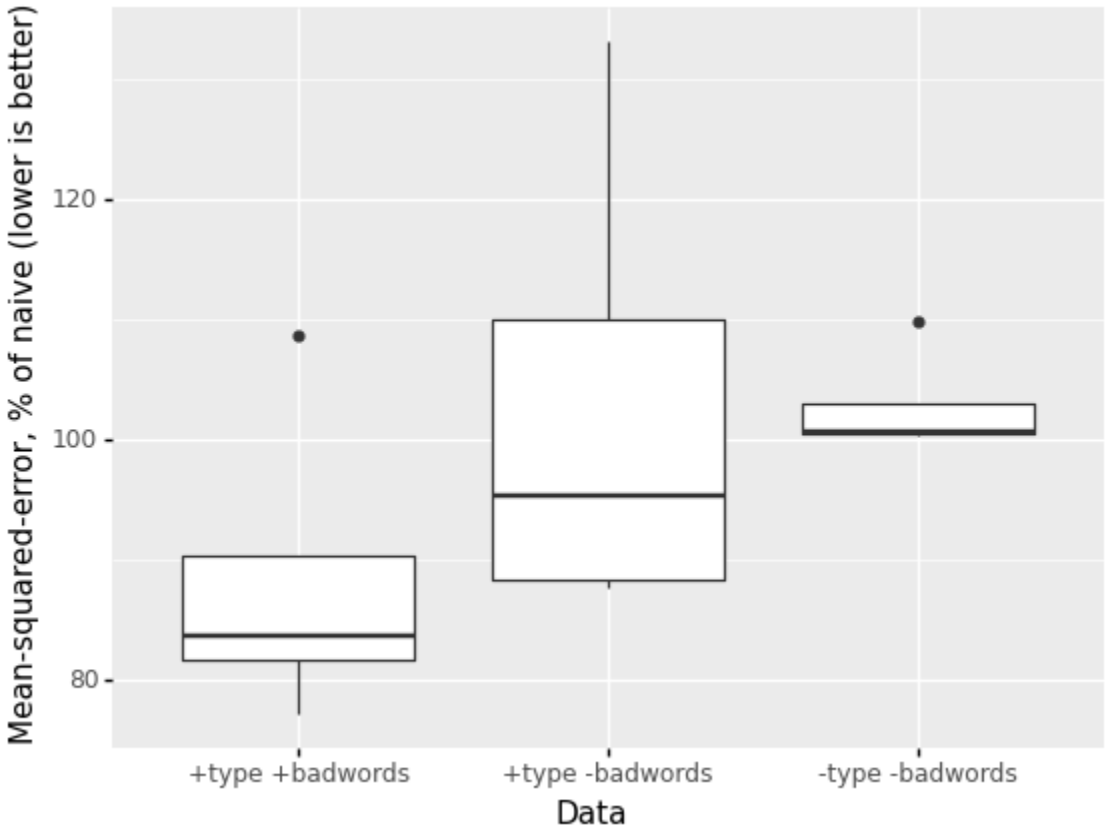
- 'destroy' in "It's not possible to destroy static types at Python exit"
- 'xx' in "The "xxlimited" module (Modules/xxlimited.c) was added as part of PEP 384 (Defining a Stable ABI), and is undocumented."

Therefore, since we can meaningfully predict disengagement with this relatively poor toxicity metric, the question isn't if we can begin to predict qualities within the sociotechnical graph, but what qualities would be deemed most valuable to predict.

Predicting instability events

Transformer-enhanced Graph Convolutional Network

In a critical first step for SocialCyber, the LAGOON team has begun predicting the instability event of contributor disengagement for the M3 milestone. We applied a transformer-enhanced variant of the Graph Convolutional Network (GCN) to leverage both the topology of the sociotechnical database (via the *type* of each entity) and the extracted *badword* toxicity information available in the prototype. This combination resulted in an 11.7% improvement in predicting contributor disengagement when compared to a naive predictor, demonstrating that both temporal topology and social toxicity information provide vital information regarding an individual contributor's stability. This is shown in the plot below. Numbers below 100% indicate effective generalization from learning to contributors for which no instability information was included in the training data. Also, outlier detection in the box plot demonstrates a greater improvement when considering the median.



The remainder of this section presents the methodology used for this result.

Data: For gathered artifacts of CPython, including the GitHub repository history and the OCEAN message list data, contributors which interacted with CPython 100 or more times were identified. This left 983 contributors as part of the data analysis. Of these contributors, 90% were reserved for training, 5% for validation, and 5% for testing; these sets were not fixed, but were shuffled for different trials to add generality to results. Time was treated as a continuous dimension, with the algorithm considering as input all activity within a single window of half a year for each evaluation. The algorithm's output was based on all activity within the following half-year window, without overlap. For each evaluation's half year input window, the full 1-hop sociotechnical subgraph around a contributor of interest would be extracted and used to predict the "disengagement value" in the next half year window. Here, the disengagement value is:

$$disengagement = min(\frac{\beta + \#commits_{output}}{\beta + \#commits_{input}} - 1,\ 0)$$

And the loss used against the disengagement is a weighted variation of mean-squared-error (MSE):

$$loss = \#commits_{input} * (predicted - disengagement)^2$$

Where β is a small value used to prevent division by zero, and $\#commits_{window}$ is the number of commits *directly from the contributor of interest* found in the specified time window. Any time frames without commits in the input window were discarded from the data set; time frames without commits in the output window were kept, as in some sense these windows indicate peak disengagement. To contextualize the measured disengagement, the significance of each data point was weighted (multiplied) by the number of commits in the input window, as shown in the loss equation. Effectively, this weights core / highly active contributors higher than spurious contributors.

For input available to the predictor, the basic graph structure within the input window was augmented with the following information at each entity's node:

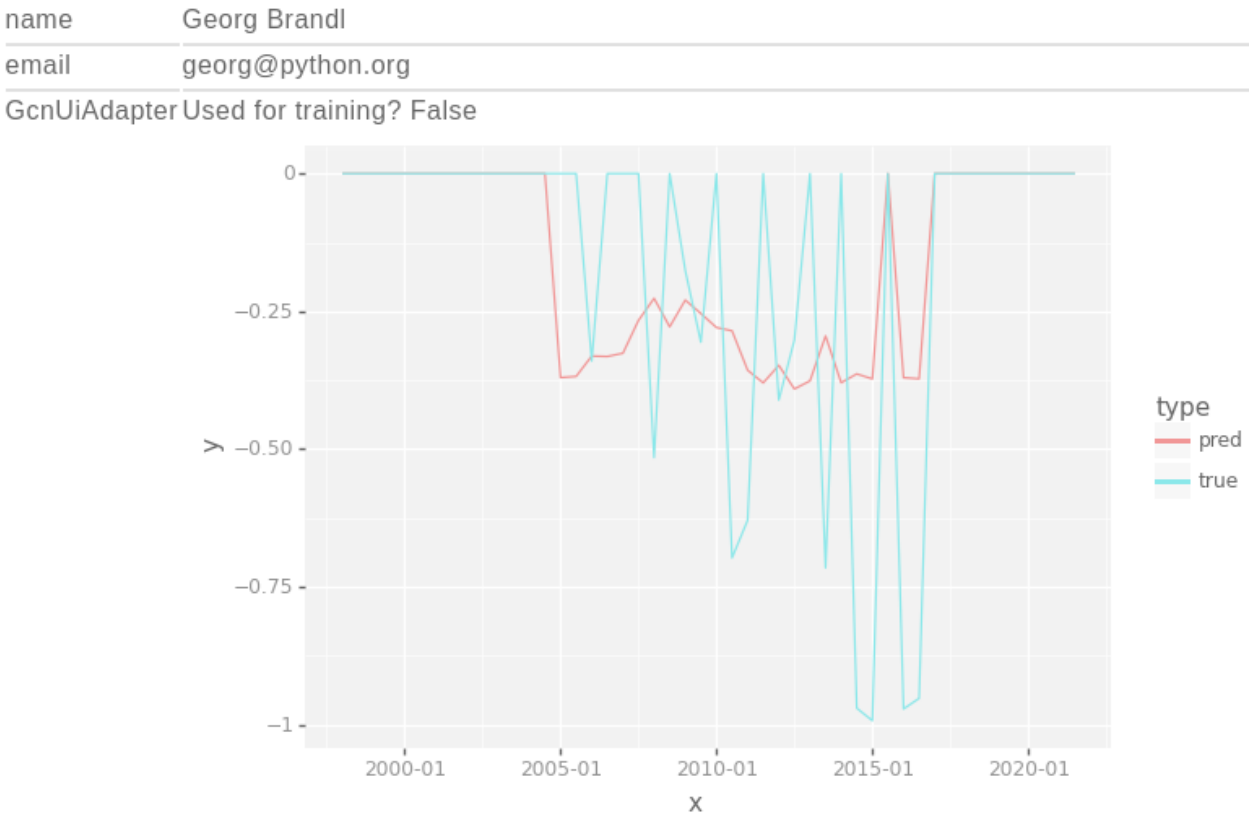
- +type: Embeddings of dimension 4 were generated, and parameterized, for each different type of entity in the database (person, git commit, message).
- +badwords: Multiple bad word lists (see Data Ingest Status) were used to create a proxy measure for toxicity. For each OCEAN mailing list message in the database, the sum of toxic words occurring in that message was attached to the corresponding node as a discriminating feature.

Scoring: For each scenario, a naive prediction was learned over 5000 samples of a 1-neighborhood around the training set of significant contributors. The naive prediction computed the weighted mean over the training set; thus, it is the best "1-parameter" predictor possible. Then, a GCN with Transformer qualities was used to predict the disengagement value. This prediction and the naive prediction were both scored in terms of mean-squared-error, and then the GCN's prediction was divided by the naive prediction's MSE, resulting in a percent improvement. This regularization step was important to calibrate for the differences in test sets

across multiple runs -- that is, the mean disengagement value across runs was often very different, effectively a result of cross-validation caused by choosing a different 5% segment of the significant contributors on each run. This was done for 4 trials for each permutation of data availability; notably, for “-type” the type embeddings were not included in available data, and for “-badwords” the toxicity information was not included.

Results: Both wordlist-based toxicity and type information were helpful, though the combination was most helpful, resulting in an average of 11.7% reduction in weighted MSE compared to the naive solution.

Additionally, the UI was extended with a means of running and displaying this prediction on any contributor in the database across the entire history of the project. The figure below is an example screenshot of the UI plugin which shows a contributor’s actual versus predicted disengagement in the CPython project at half-year intervals. The UI indicates that this contributor was not part of the training data; that is, it is a test case, which demonstrates the generalization capabilities of the prediction system. Notably, it correctly inferred that the contributor was unlikely to become disengaged between 2007 and 2010.



Notably, we did try a version which used the 2-hop neighborhood around the contributor. This 2-hop variant was prone to over-fitting, and did not predict disengagement as successfully in a generalizable way.

Graph-based Multi-layer perceptron (G-MLP)

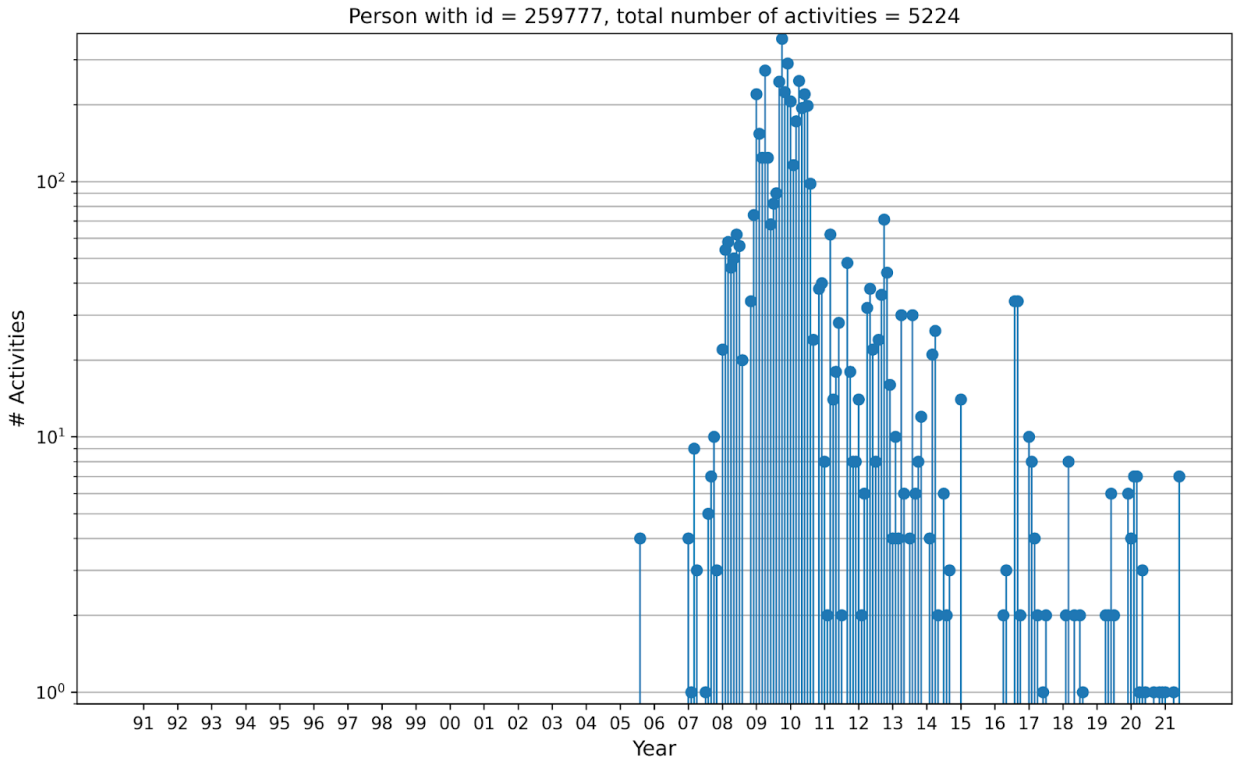
We also implemented an approach based on extracting features from the graph and applying a multi-layer perceptron (MLP) neural network to predict contributor disengagement. This approach currently results in a 6% improvement in predicting contributor activity disengagement as compared to a naive predictor, with greater improvements possible by more extensive exploration of the hyperparameter space.

The G-MLP approach temporally slices the graph into two-year windows with an overlap of one year between adjacent windows. Toxicity features in the form of wordlist-based bad word counts are extracted from a particular window, and used to predict contributor disengagement on the next window. The features are summed over from all nodes one and two hops out from each contributor. One significant source of improvement that we have already implemented is processing the data to remove contributors who have no toxicity in any of these surrounding nodes.

Disengagement is measured in two ways:

- Activity, which is the fractional reduction in a contributor's activity from one window to the next;
- Gaps, which is the number of 'lean periods' for a contributor in a window, where a lean period is defined as a period of 6 months or more without any activity.

The figure below is an example of 'gaps' disengagement computed from the graph for a particular contributor. Each vertical bar is the number of commits in a month (log scale). This contributor became active in the middle of 2005 with 4 activities, then (potentially suspiciously) did not engage until 2007. S/he was then active regularly until the end of 2014, following which there were lean periods in 2015-16, the latter half of 2017, late 2018 to early 2019, and late 2019. Thus, taking 2017-2018 as an example window, the Gaps count is 2.



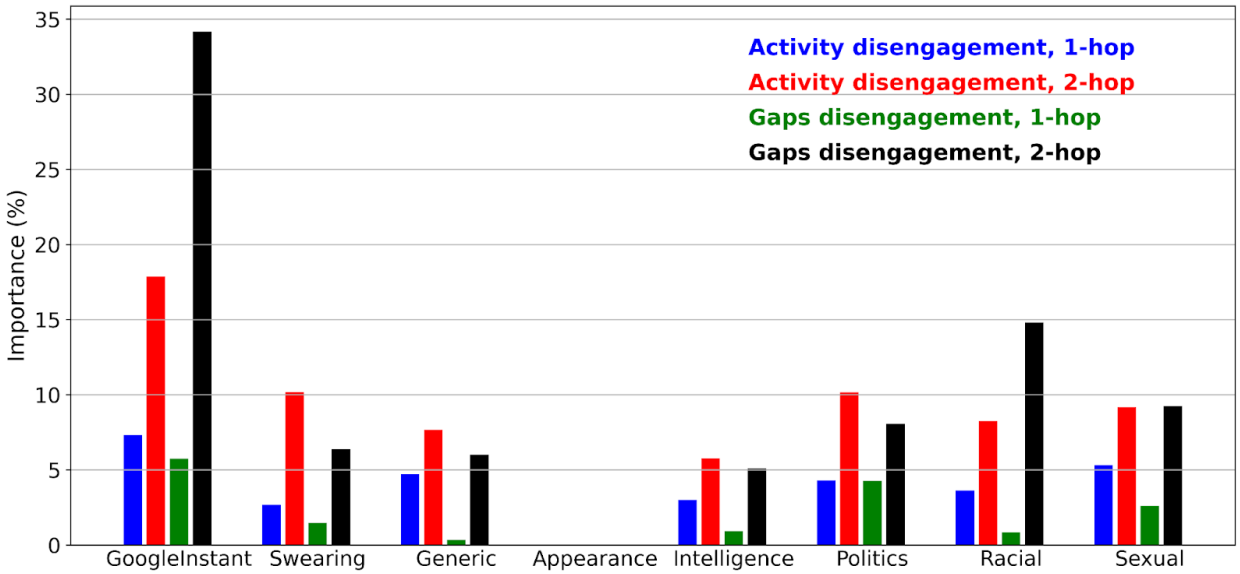
Other approaches

We are currently implementing an approach based on GraphSage (Hamilton et al. 2017 https://arxiv.org/abs/1706.02216) – a popular method for performing machine learning on graphs that scales well to large graphs which have nodes and edges added with time, as is our case.

Ensemble machine learning methods such as gradient boosting and random forests are less effective (4% improvement over a naive predictor) than the neural network methods discussed thus far, however, they are excellent tools for predicting feature importance as described in the next section.

Feature importance for prediction

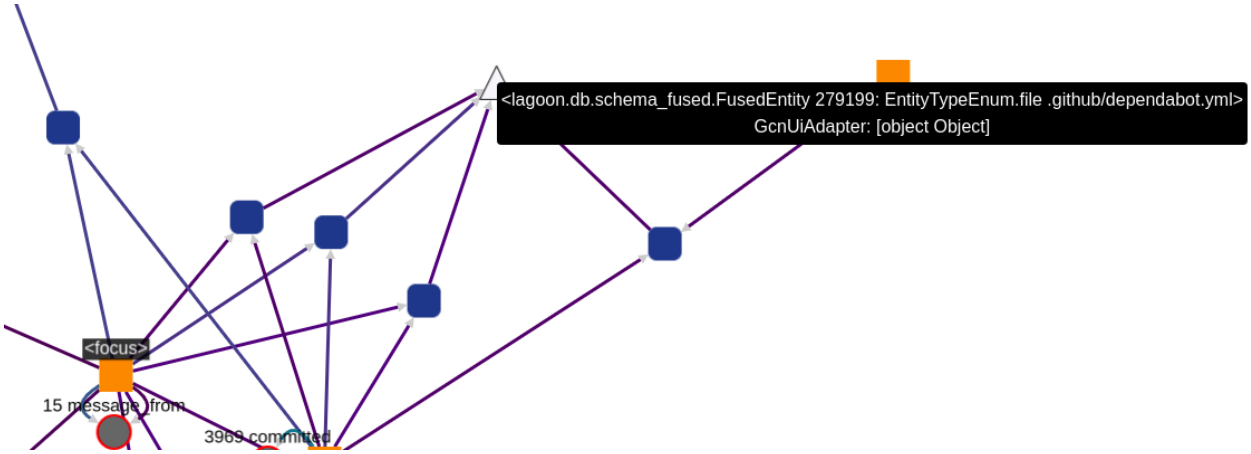
As mentioned previously, the foremost way to improve prediction capabilities for instability events will be to add better features to the data available to the predictor. Keeping this in mind, we attempted to assess the importance of the different existing features using ensemble machine learning methods. The features considered are 8 categories of bad words, each summed over all the messages one and two hops out from contributors. The categories are as mentioned before – words banned from ‘Google Instant’, a custom wordlist ‘Swearing’, and lists of toxic words categorized into 'Generic', 'Appearance', 'Intelligence', 'Politics', 'Racial' and 'Sexual'. The importance was assessed from the results of the top 10% of gradient boosting models.



The percentage importance results shown above indicate that a count of the list of words banned from the Google Instant platform is a feature of high importance when predicting instability. In particular, aggregating this information from the larger group of messages 2-hops away from a contributor is the foremost indicator of disengagement. Its effect is even more prominent for Gaps disengagement as compared to Activity disengagement. Racially and politically charged bad words are also useful features. On the flipside, bad words in the category of ‘Appearance’ have negligible importance.

Identifying shared code segments

Characterization of the social history and technical ownership within the project is currently an ad hoc process which can be managed and understood through the UI. Since the UI leverages windows of time to filter and orient the analyst’s attention, one can search for a particular file, such as .github/dependabot.yml. By looking at the spatiotemporal graph around this point, we can see that this file has been modified by two contributors, Mariatta and John Losito (orange squares). These contributors modified the file across 4 commits spanning October 2020 to February 2021. We can also see that John Losito only created 1 of the commits, with Mariatta creating the other 3 and committing all 4. Thus, Mariatta owns this particular section of the CPython code, a deduction which can be drawn with a very small amount of interaction with the UI.


<lagoon.db.schema_fused.FusedEntity 279199: EntityTypeEnum.file .github/dependabot.yml>
GcnUiAdapter: [object Object]
<focus>
15 message_from
3969 committed