 # 1. Begin with an introduction/abstract to your research and topic area. This should be a maximum of 400 words.

My research is in web application and IoT platform security. This research topic falls more generally within the software or application security areas. Currently I am researching how to improve vulnerability detection for stored XSS faults in web applications.

Injection faults, such as cross-site scripting (XSS), remain one of the most prevalent security flaws in the modern web, despite being both relatively simple and known for over 20 years. These faults are typically found with automated analysis, known in this context as vulnerability detection. Vulnerability detection is usually implemented either by white-box static analysis, or black-box dynamic analysis. White-box static analysis is limited in its technical specificity to certain languages or technologies, its ability to accurately model dynamic behaviour (in a performant manner), and its usability requiring code access. In contrast, black-box vulnerability scanners do not suffer from these limitations, but also lack in their performance. Black-box scanners are still unable to find many injection flaws due to intrinsic difficulties with crawling. Augmenting black-box crawlers with some additional access to resources, to aid their understanding of the application they are analysing, while still not resorting to static analysis of code, might help improve their performance.

I don't work with either machine learning or artificial intelligence directly; my relation to WASP can be understood as contributing to an enabling technology for eventual ML and AI deployment; a secure web and IoT ecosystem.

 # 2. Select at least 2 principles/ideas/concepts/techniques from Robert's lectures and discuss how they relate to your research and topic area.

Behavioural software engineering

Behavioural software engineering's premise seems useful when applied to the broad area of cyber security. Just as software engineering can benefit from focusing on humans, rather than technology or process, so can security also benefit. After all, an unusable system is perfect from a security sense (as no one can or will use it, and it will not be compromised), but also perfectly useless. Therefore, this human focus is necessary to have any real notion of utility.

In my research, I must make sure to consider the utility of any method I propose, and especially think about the people that will (hopefully) eventually use them. Related fields to cyber security, such as computer privacy and usable security, focus even more on this human aspect. Instead of focusing on securing a system from an attacker, researchers in these fields might instead focus on improving the understanding and (real) consent a user has with these systems. These limits on data inflow can lead to preventing these systems from having sensitive information in the first place, making their compromise less damaging. This more preventative approach, where the damage insecure systems can cause is potentially minimised, is complementary to securing the same systems.

A concrete example of needing to consider usability, from my previous research into IoT platform security, is the problem of defining access control policies. Static and dynamic monitors can both provide reasonably strong guarantees about an application's execution, beyond the state-of-the-art used in industry, given some access control policy defining what sensitive APIs, information, etc can be accessed by which users. However, these monitors cannot be easily applied until the policies they enforce can be easily defined, extended, modified, and understood. This remains an open research question.

Pragmatic SE rules

Just as Software Engineering has defined some practical rules, so has security. However, to my knowledge, these rules are not as widely known across subfields, and have not been compiled into easily accessible volumes.

Software testing

Verification VS validation

Software testing also has an obvious relation to vulnerability detection, as it can be seen as a specific application. I would say that vulnerability detection tools are performing validation, as they check that an implementation meets some security goals, which are not typically encoded as requirements or specifications. Instead, they are defined by some attacker model, which I argue definitely falls under the 'problem' aspect of 'problems and needs'.

Other forms of validation in security, that might have to stretch the definition less, would include formal methods or cryptography proving that some protocol has particular security guarantees.

Static VS dynamic (and black-box vs white-box)

Both static and dynamic testing techniques are also employed in security. As in software engineering, they have their own advantages and disadvantages, and are best employed in conjunction. Static analysis is limited by code availability, almost guaranteed for analysis of malware or other malicious code. Therefore, subfields of malware analysis that approach problems such as renaming, retyping, and other forms of reconstructing source code from compiled executables, exist.

Huge input/output spaces

Huge output and input spaces are problematic for web vulnerability testing. In fact, in black box settings, these will usually be infinite. Without some understanding of the web application, this will be the case for any dynamically generated content. For example, when crawling a website, an interactive crawler could make new posts, which then expose new comments, and so on. Heuristics combined with modelling and search techniques help to alleviate this. In the previous example, a scanner that models the application and finds similarities between post and comment states should be able to collapse these to one state in the scanner's internal model, and stop infinitely wasting effort exercising that single functionality.

3. Select at least 2 principles/ideas/concepts/techniques from one of the three guest lectures and discuss how they relate to your research and topic area.

Felix Dobslaw - Boundary Value Testing for Traditional and ML Software

Felix's guest lecture on exploring boundary values has some pertinence to software security. Boundaries between secure and insecure behavior, in that they define what makes some particular attack/payload work (incorrectly/insecurely), are exactly what vulnerability detection attempts to find, as they would provide both an example exploit, but also explain it by that boundary. Boundary value analysis (BVA), or analyzing artifacts to determine expected and actual boundaries, is not necessarily useful, as I would expect insecure application behavior to just as likely result from unrelated functionality or undefined behavior.

One instance where BVA, in the form of boundary value exploration (BVE), might nonetheless be applicable is in aiding the analysis of permission models. These permission models can be very complex; web applications such as WordPress have both predefined roles and dozens of contained capabilities. Manual analysis is tedious, time-consuming, and error-prone. Visualizing boundaries to identify visual boundary candidates would be helpful, though I'm not sure how this could automatically be performed. ML techniques like eigenvectors don't seem readily applicable.

WordPress roles and capabilities:
https://wordpress.org/documentation/article/roles-and-capabilities/

DevBots AI/ML support for SE - Linda Erlenhov

Linda's guest lecture on ML/AI4SE, rather than SE4ML/AI was also quite interesting, though less directly relatable to my research or area.

One interesting point is that ML/AI4SE seems like it would be both an application that needs security, as it deals with confidential data in the form of software development (could be used to compromise industrial secrets, extract vulnerabilities, etc), while at the same time being hard to secure (due to the ML models and lack of explainability, robust security controls).

Emerging areas, such as LLM security, can play a role in securing ML/AI4SE. However, their results are so far quite negative. Existing mechanisms, such as guardrails, have been easily bypassed. Furthermore, these attacks (jailbreaks) can be generated automatically, and are not specific to any one system, but instead utilize fundamental weakness.

LLM guardrails fundamentally insufficient:
https://arstechnica.com/ai/2023/08/researchers-figure-out-how-to-make-ai-misbehave-serve-up-prohibited-content/
https://llm-attacks.org

 # 4. Choose two topics from the following list:
Automated Software Testing
Security and Privacy

For each of the two topics you have selected, first briefly discuss your understanding of them. Then describe areas of opportunity (new research challenges, commercial opportunities, application of Software Engineering ideas, methods and tools in AI/ML) with regard to these topics and your area of interest/research (in your PhD project). This should be approximately 1.5 A4 pages in length. For each topic, search for and find at least one recent (published in the last 4 years) paper to inform your writing and thinking about the topic and its connection to your own research. As much as possible, relate the topics to your own experience and research / project.

Automated Software Testing

Fuzzing, where a program is tested with generated inputs with some random mutations, has been used to improve software testing performance since its inception. Fuzzing's first applications were in testing Unix utilities for correctness (if these utilities would crash under some input). Many flaws were uncovered, and subsequent evaluations still find many more. In the security community, fuzzing has been used to test all manners of software: compilers, libraries, OS components, etc.

Some of these are highlighted in 'The Art, Science, and Engineering of Fuzzing: A Survey' from 2021. Authors note that while fuzzing is increasingly used, researchers and practitioners have not formed a consensus towards a unified view of fuzzing. Authors believe fragmentation will make it hard to learn about fuzzing or improve it, and hinder the field's progress in the future.

The systematization and taxonomy put forward in this paper, along with the categorization of fuzzing papers using this systematization and taxonomy are both useful results. I will have to examine these results more carefully in the future, as I believe they are very relevant to my own research.

Vulnerability scanners do something like fuzzing, where they usually are mutating their 'input', either in location, or in the content of input itself. I haven't used fuzzing directly, as in applying the standard

tools and developing fuzzing harnesses for my specific application, but it would be very interesting to see how a vulnerability scanner could be implemented in such a way. This very well can lead to performance improvements (detecting novel vulnerabilities), due to borrowing other developments in the very active fuzzing community.

'The Art, Science, and Engineering of Fuzzing: A Survey':
https://ieeexplore.ieee.org/abstract/document/8863940

Security and Privacy

Security and Privacy help to protect the users of software, either against attackers in the case of security, or against both attackers and software owners in the case of privacy. Security deals with things like attacks (vulnerability, exploits) and defenses on CIA (confidentiality, integrity, availability), while privacy deals with consent, minimizing data flows, and integrations with usability, legality, etc.

In the context of Software Engineering, the integration of security and privacy into the software development process or lifecycle seems like an interesting, though challenging area. The paper I choose is 'Integrating security and privacy in software development' from 2019.

Authors contribution is primarily in addressing privacy, despite the title. According to the authors, privacy is a more open question than security, as they believe security's integration to the SDLC to already be sufficiently addressed. They propose 'Privacy Oriented Software Development' (POSD) and a 'Privacy Knowledge Base' (PKB) to guide it.

I don't agree with the summary of the field the Authors perform in section 2. They state that 'The concepts of security and privacy in software development are strictly related to each other'. I don't think this is true - a private platform can be insecure, and a secure platform can completely ignore privacy issues. Security and privacy are complementary and can help to assure that each other's goals are reached, but they are not strictly related. Author's characterization of existing work also highlights classic manual work such as penetration testing as how security is integrated into SE, which I don't think is sufficient with modern SDLC. Reports need to be fast and automated.

PKB is based on five concepts, 'Principles of Privacy by Design, Privacy Design Strategies, Privacy Patterns, Vulnerabilities, and Context.' This conflates security and privacy, in that the PKB somehow maps vulnerabilities to privacy design principles violated. Privacy does not deal with vulnerabilities, though they are right to highlight 'privacy design', as this is a correct link.

Regardless, the principles they propose make sense for privacy (at least), as are the strategies to achieve them, and patterns with which to implement/include them.

However, this falls off again, as their detailing of vulnerabilities is both simplistic and inapplicable to privacy goals.
Finally, their description of the context of privacy-oriented SD in a server-client architecture is good, though fundamentally flawed in that it is designed around attack scenarios and vulnerabilities, which can improve security, but do not make sense with privacy. The rest of the paper has the same flaw.

In the context of my research, integrating research results, such as vulnerability detection tools, into the real world is very interesting. I think this can be improved. I'm especially interested into how this could be achieved within continual deployment; new requirements on automatic vulnerability detection such as stricter runtime performance (faster, lighter) will have to be met.

'Integrating security and privacy in software development':
https://link.springer.com/article/10.1007/s11219-020-09501-6

 # 5. Discuss your thoughts regarding the future trends and directions of Software Engineering in relation to your topic, your career (either in academia or in industry), and AI/ML in general. You should at least discuss ML/AI Engineering and how you think it and its importance will change in the coming 5-10 years.

ML/AI Engineering - SE4AI/ML

I think the fourth lecture on ML engineering and requirements engineering presents some particular areas of AI/ML that will become more standardizing, and fall more within Software Engineering, within the near future. Without this standardization of the SDLC for AI/ML, prototypes will not fulfil requirements, both in development and deployment. These requirements are also not known, as AI/ML has different (especially non-functional) requirements than traditional software. Unless both requirements are well-defined, and AI/ML follows some standardized SDLC, AI/ML products are liable to be unreliable and possibly expensive. AI/ML products will either not be adopted in industry, or they will be used irresponsibly. Either case is bad, hence why I believe this must change in the near future.

I am not an ML/AI practitioner, so perhaps my belief in the necessity driving this change is unfounded and I am naive to believe anything will happen in a reasonable timeframe. However, as a security practioner, I can imagine the failure's possible consequences.