如何在计算机应用领域寻找研究想法

钱志云
计算机教授@加州大学河滨分校

How to look for a research idea

https://zhuanlan.zhihu.com/p/341685279?utm_source=wechat_session&utm_medium=social&utm_oi=42975992217600

Over the years, I have mentored and collaborated with over a dozen Ph.D. students and found a common set of challenges that prevent them from progressing better in research. This article is my attempt to offer some tips as guidance to Ph.D. students in computer science. My research area is cybersecurity and hence students in this specific area may find it more relatable due to the examples I will give (but it should also generalize to other practical areas such as networking, systems, and architecture). In case you are interested in what I do, you can go to my homepage. The opinions in this article are mostly based on my personal experiences and therefore can be biased. Nevertheless, I hope some of you might find it useful. Here we go:

For those of you who have never done any research before getting into a Ph.D. program, it can be scary to even begin thinking about where to start. There are two ends of a spectrum: (1) You are fortunate (or unfortunate) to be in a super productive group where a lot of ideas are thrown at you --- this happens often when you work with a junior faculty (of course with exceptions). (2) You are in a well-established group where your advisor no longer gives students concrete ideas to work on. Instead, what they might do is to give a very high-level direction along with a few related papers. That is all you have to work with to come up with a research idea.

If you are in the first type of group, it is obviously great to get your first research project going and walk through the complete process ... except you are not actually going through the entire process from end to end. In fact, you are bypassing, in my opinion, the most difficult step of a research project --- coming up with a good research idea. Everything has a tradeoff. Yes, you do get an earlier publication and possibly a very good one because of the idea selection and formulation by your advisor (or a postdoc or senior grad student). But, you lose the opportunity to get trained to find your own ideas independently, which is a critical skill of a Ph.D. I find myself guilty as an advisor too, e.g., by doing the hard part of discovering a vulnerability and having the student execute the rest.

If you are in the second type of group, it can be a make or break situation. Either you manage to figure out a good idea to work on, or waste the first one or two years of your Ph.D., realizing it is not going anywhere and deciding to quit. When I was a Ph.D. student, I have witnessed about a

50% drop out rate in groups that are primarily the second type. In either type of group (or anywhere in between), you want to start the training process of looking for a research idea as early as possible. Else, I consider it a big deficiency and may hurt you significantly in the future after you graduate, e.g., you may have a hard time leading a research program independently as a professor in academia or researcher in industry. Below are a few tips I find useful:

Tip 1: Learn to read papers and develop your taste

Ideas don't come out of the blue. One of the most common ways to generate new ideas is by reading other papers and getting inspired. Pay special attention to the relevant seminar classes where you begin to read a ton of papers. Most likely you are also required to write paper responses to express your opinions, critiques, and any constructive thoughts. It can be initially overwhelming to read 3-4 papers each week per class (I was taking two such classes in the same semester when I was a student). Hang in there. One common misconception is you have to understand *all* the technical details of a paper to consider the job done. No, that is neither the main goal nor an effective use of your time! In such classes, it is really a process of learning to appreciate and criticize research ideas, e.g., why is a paper good or bad? What makes an interesting paper? You don't have to read every single detail of the paper to answer these questions. There are actually many helpful articles about how to read research papers. For example, How to Read a Paper by S. Keshav.

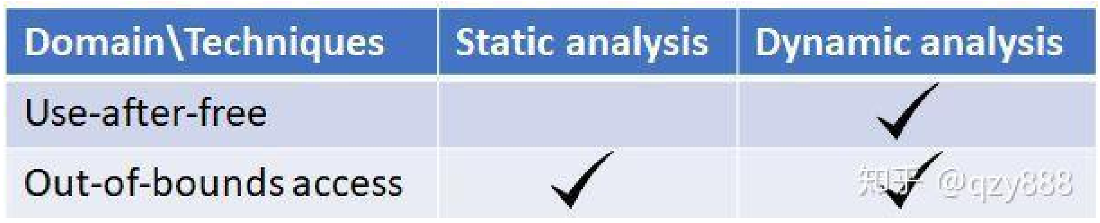
More importantly, find what types of papers interest you the most and ask yourself why. Take the cybersecurity area as an example, it is so broad that I may not even be able to capture everything. In terms of research domains, any CS field such as operating systems, network protocols, or any software and hardware, has its corresponding "game rules" and "security threats" that can be researched. And human beings, which are often the weakest link, also play an important role in cybersecurity. In terms of research styles, they range from novel attacks and exploitation techniques, analysis of emerging systems or an algorithm, to defenses, measurements, and more. Another dimension is the research techniques used in a paper: manual analysis, reverse engineering, program analysis, formal methods, hardware-based system design, data-driven approaches such as machine learning and AI. Finding your favorite types of papers will help you develop your own research taste, and ultimately narrow down your scope in forming a new research idea.

In the early stage of my career, I was fascinated by novel attacks that break state-of-the-art defenses. They are creative, elegant, and great satisfaction because you are able to discover security flaws that no one else sees. When I read papers like these, I always ask myself "How did these people manage to find the flaws? What skills are needed to conduct such research?" This has consciously or unconsciously led me to develop the necessary mindset and skills for my Ph.D. following this style of research. Since my advisor focused on networking at the time, I naturally worked on security problems in the networking domain (e.g., TCP security). However, at the time I did not get a chance to acquire any solid techniques such as program analysis, formal methods, and machine learning. When I was about to graduate, my research taste started shifting accordingly, as I realized that it is not really sustainable or scalable in solving security problems without proper automated techniques in my toolbox moving forward. As a result, I started to learn program analysis, and a bit of model checking and machine learning / AI. This has worked out great for me personally. Of course, everyone is different and you can always find your own taste and develop a unique path accordingly. In fact, this is exactly the beauty of academic freedom!

Tip 2: Recognize patterns of developing research ideas

Every paper has a unique story behind it, but the way they come about can often be classified in a few patterns. Most of these are distilled from my own research experience and some are borrowed from other researchers. For example, Prof. Philip Guo from UCSD had a great blog post (it was said to be taken down but you can still find here) about the various patterns which resonate a lot with me (can't remember all the details at this point). Below is my version with concrete examples to illustrate them. Note these patterns may not be completely orthogonal. They are simply ways to realize how an idea comes about. Once you master a few of these patterns, it will be a piece of cake to come up with new ideas (of course you still need to triage the value of an idea before deciding to commit to it).

Pattern #1: fill in the blank



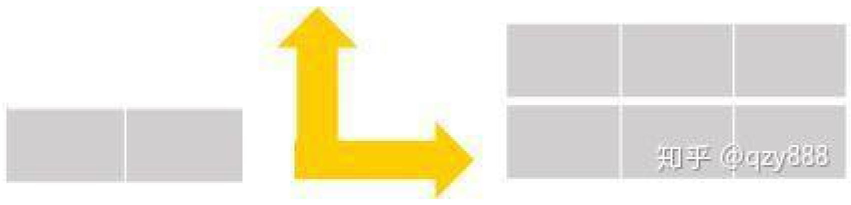
Domain\Techniques	Static analysis	Dynamic analysis
Use-after-free		✓
Out-of-bounds access	✓	✓

This is a simple pattern that I heard about when Dr. Harry Shum from Microsoft gave a talk about research during my undergrad years. The idea is very simple conceptually. Read a few papers on a topic, jot down some notes about the differences among these papers in terms of assumptions, guarantees/properties offered by a system, methodologies, techniques, datasets, etc. and then draw a table (not necessarily 2-dimensional). Look for empty spots and those are potential new research ideas you can work on (in fact many good papers use such a table or figure to clearly distinguish themselves from related works). One example is in the space of adversarial machine learning where many researchers have initially focused on images, e.g., how to perturb a picture of a cat so that it is misclassified as a banana. Soon other domains such as video, audio, and malware are investigated (e.g., how to fool a malware classifier by injecting meaningless instructions). Another simple example is: people may have applied static analysis to automatically find certain types of vulnerabilities but no one has applied dynamic analysis yet. You can then investigate the pros and cons of static vs. dynamic analysis on specific types of vulnerabilities. Of course, in reality, there are also various types of static and dynamic analysis. You can make a much more fine-grained table which will give you more opportunities to identify the blanks. Similarly, if a certain type of vulnerability has been studied, you can potentially work on a different type.

The key to applying this pattern successfully is to map out the dimensions that exist in a space. The deeper you can go, the more likely you will find blanks to fill. For example, you have to be familiar with the different kinds of program analysis techniques before you can draw a table and convince yourself that a specific technique has not been tried out to solve a problem. Similarly, you need to be aware of the different kinds of vulnerabilities (there are a dozen of memory corruption vulnerability types alone).

There are generally two common ways to map out the dimensions. First, you can read broadly and look for differences between papers on similar topics. Trust me, they will become handy one day when you connect the dots. Another way is to read survey papers can potentially be a shortcut as it typically summarizes a space in a number of dimensions already. In the cybersecurity space, the IEEE Security and Privacy conference (a top one) accepts and publishes a few papers each year in the form of Systemization of Knowledge (SoK). They are definitely worth reading if the topic interests you.

Pattern #2: expansion



This is a natural progression of “fill in the blank”. As I mentioned, mapping out some good dimensions of a space can be the most challenging step there. However, if you already have some ideas in a space (e.g., published a paper or two), you can be in an advantageous position to see dimensions that are not obvious to others.

Our USENIX Security 16 paper on a strong TCP side-channel attack (Off-Path TCP Exploits: Global Rate Limit Considered Dangerous) was motivated by this pattern. In our prior work (published in S&P 2012 and CCS 2012), the attack requirements were very high (and unrealistic, one could argue) where a piece of unprivileged malware is assumed to have been installed already on a victim smartphone (or a certain type of firewall is deployed in the network). Naturally, I have been thinking hard to eliminate such strong requirements and it leads to the discovery of a brand new side channel. In essence, I have basically mapped out the dimension about attack requirements and can be visualized as [Requirements : Malware | Firewall | Non].

Another example of our CCS 20 paper “DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels” also falls under this category. This is another side channel paper where we transferred our side channel experience from TCP to UDP. The nature of the vulnerability is very much similar. In fact, it was along the same exact direction of looking for shared resources (i.e., global variables) across TCP/UDP sockets in the Linux kernel implementation. So the dimension here is about the different types of network protocols.

Pattern #3: build a hammer and find nails

This was also learned in the same talk Harry gave and I generalize it a bit here. The high-level idea is that if you have a unique expertise, technique, system, or even dataset (that no one else can easily replicate), you can take advantage of it and use it to look for interesting problems to solve (we are lucky to have so many practical problems in computer science). One classic example in systems research I can immediately relate to is Prof. Peter Chen’s group at the University of Michigan. His group built a strong expertise in virtual machines (among other things) and leveraged it to develop many interesting applications on top. If I remember correctly, Pete’s group built the world’s first full virtual machine record and replay capability. This has important applications for debugging, retroactive intrusion analysis. A number of top-quality papers (e.g., OSDI, ASPLOS, PLDI) are published on this topic with regard to determinism guarantees, performance, etc.

My own group has also published a series of papers in network side channels, initially from TCP back in 2012, to UDP and DNS now in 2020 (7 papers in S&P, USENIX Security, and CCS). It is really about building up the expertise in network side channels, which is admittedly a small area without too much competition. The nice thing about building up your expertise this way is that once you are so deep into a topic, it is much easier for you to find new problems to solve.

A recent example in cybersecurity is the system developed by UCSB folks called angr. It is a state-of-the-art binary analysis framework (including a symbolic execution engine) that was originally developed for the DARPA Cyber Grand Challenge (a 100% automated CTF competition). The system was open-sourced since 2016 when the paper was published. Because it is so well designed and engineered, it quickly became popular among both academia (540 citations already on 12/28/2020) and industry. In fact, we were using it in a few projects as well. The authors themselves have also leveraged the system to build a number of follow-up projects for various security applications.

One last example regarding datasets is the Center for Applied Internet Data Analysis (CAIDA) at UCSD where they developed and deployed several measurement infrastructures to collect a vast amount of Internet data for various purposes. Since no one else has the data (in academia at least), they were able to publish many interesting measurement studies with the large-scale and unique datasets.

Even though this is a great way of conducting research in my opinion, i.e., impactful, sustainable, it is not necessarily for everyone. First of all, it can be extremely time-consuming to build the expertise, system, or infrastructure to the point where you can start reaping the benefits. Secondly, such initiatives can require a vision, planning, and organization of a team sometimes (often beyond a single Ph.D. student's ability). Lastly, few groups typically dominate a space. It is very difficult to surpass them unless you have a unique perspective. That said, if you are able to identify something where a lot of people need it but no good solution exists yet, it might be worth considering.

Pattern #4: start small, then generalize

Oftentimes a research idea starts with a small observation. After some digging, you can see if it can be developed into a full idea worth publishing. Now, it can be tricky to determine whether to pursue it and how much time you should spend digging before giving up. There are in my experience a few positive signs that you can look for: 1. The initial observation (however small) is so intriguing and surprising when you first saw it. 2. When you dig deeper, you realize that the phenomenon is rooted in something fundamental and cannot be explained easily by well-known concepts (e.g., a novel design flaw that leads to security vulnerabilities). 3. The observation is unlikely to be one-off, e.g., there is a bigger space behind and there are many other similar situations that you can investigate.

A concrete example is our own work published in CCS 16 “Android ION Hazard: the Curse of Customizable Memory Management System”. We were initially poking around different interfaces exposed through file systems to any potentially malicious app on Android. My student Hang stumbled upon a device file “/dev/ion” which looked interesting. The interface allowed any app to allocate memory in “pre-existing heaps” and map them into user-space. Surprisingly, we find that instead of returning “zeroed pages”, these pages contain leftover data from prior use (sign #1). This is a potential information leakage security vulnerability. Now, that by itself may not lead to a direct research project because it may simply be viewed as yet another instance of a well-understood information leakage vulnerability. However, when we dig deeper, things started to fall in place. Even though the bug type is not new, the flaw is more deeply rooted than what it initially appears to be (sign #2). Specifically, the introduction of the /dev/ion interface has unexpectedly exposed memory returned by internal APIs used by OS kernels. In contrast to certain APIs that are designed to interface with user-space (where zeroing is a requirement), such internal APIs do not zero newly allocated memory proactively because of performance reasons. Even worse, different Android smartphones customize the implementation of /dev/ion, allowing a bigger space of investigation (sign #3).

Pattern #5: reproduction of prior work

The previous pattern may get you to think: how do I make these small observations or discoveries in the first place? Well, one of the ways is to try and reproduce the result of a published paper. Believe it or not, what was reported in a paper may not be exactly what you will observe when you try to reproduce it. There are a few reasons behind it: (1) inadvertent mistakes made by the authors, (2) some results being not 100% reproducible by design, e.g., measurement of some Internet phenomenon which changes over time. (3) biased benchmarks or datasets picked which favored a method proposed in a paper. If you are able to identify important discrepancies or limitations of prior work, it often means room for improvement. Even if you manage to reproduce the work 100% as expected, there can often be other insights or side discoveries that were not mentioned in the paper along the way (there is only so much space in a paper).

I haven't personally practiced this pattern much. The closest example is our USENIX Security 20 paper “Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices", a collaborative effort led by Tsinghua University. In this paper, the students were asked to reproduce the CCS 18 work “Domain Validation++ For MitM-Resilient PKI” where a particular measurement was performed to showcase the wide applicability of an attack method. Interestingly, the measurement result performed in our paper is a lot more negative than what was reported. This has prompted a new method to be developed, which can bypass the limitation of the prior work.

Pattern #6: external sources: from industry, news feed, etc.

If you work in a practical area such as cybersecurity, there must be a lot going on in the industry and the real-world (outside of academia). Take opportunities to connect to industry people, learn their pressing needs and pain points. They are great sources of research ideas. Even though more resourceful than academia, the industry has a different prioritization and mindset when it comes to solving technical problems, e.g., they are more interested in reliable solutions and less likely to invest in risky ones. The nice thing about academia is that we don't have to solve the problem 100% in one shot and it can be exploratory in nature. In fact, it is in some sense up to you to define a metric/threshold of success. If you are solving a real problem in the industry where no good solution exists yet, the bar for a "successful" research project is considerably lowered.

I have personally taken advantage of this pattern often and enjoyed it very much. I will use one recent example to illustrate the pattern and it is about "patching". "Why is patching an interesting research problem?" you might ask. Well, through talking to folks in the industry a few years back, I became aware that it is still a huge problem in the industry and no real good solution exists yet. Here is what happens in Linux and Android kernels: when a patch is committed to the upstream Linux kernel (i.e., mainline), maintainers of downstream kernel branches such as Linux LTS, Ubuntu, and Android need to manually check if these patches "apply" to them. If so, they backport it. This is a time-consuming and error-prone process. Oftentimes, important security patches are missed or delayed because of this. Even worse, besides the owners of the downstream kernel branches, it can be difficult for third parties to audit them and look for missing patches. In the Android world, for example, most vendors do not provide a complete history of their kernel source (e.g., a git repository). This has led to the development of an automated tool that can test the presence of a patch in a binary kernel. The work was published in USENIX Security 18 "Precise and Accurate Patch Presence Test for Binaries". Subsequently, we have perfected the tool and published another paper in USENIX Security 21 "An Investigation of the Android Kernel Patch Ecosystem" further investigating the delays and root causes of slow patch propagation. In fact, the patching project has opened the door to a number of related projects that are cooking.

Sometimes, paying attention to news feeds and things that are happening around you can pay dividends too! I learned this from a few professors who regularly use Twitter as their tech news feed. One interesting project we did was inspired this way (the paper was published in CCS 15 "Android Root and its Providers: A Double-Edged Sword"). Back then, it was fashionable to "root" Android phones so that users can customize the OS and unlock new features that were not possible otherwise. A number of "one-click root apps" were developed that can automatically root many phone models with literally a single click of a button. After realizing these apps are basically launching an attack against the OS kernel, I started to think "how many different exploits do they have that can target so many phones? Do they have any proprietary exploits that you can't find on the Internet? Is it possible for an attacker to steal these exploits and repurpose them (e.g., to build ransomware)?" It turns out some of these apps are developed by top hackers employed in the industry and contain over a hundred exploits, and with some work (of reverse engineering), one can own them all (yes, pretty scary, huh).

Other patterns specific to cybersecurity research:

1. Adversarial research. Since security is inherently adversarial, a recurring theme is attacks and defenses. You can always attempt to break an existing defense or build a defense against an attack. In fact, I have often seen a novel attack paper followed by a defense paper (which may or may not come from the same group).

2. Automating a process. Much of the systems security analysis such as reverse engineering, vulnerability discovery, bug triage, exploitation, and checking if a patch is applied has always required some manual effort, at least in some settings. Automating such a process (even if partially) applying techniques such as program analysis can have significant research value. This may be a pattern that goes beyond cybersecurity but is very common in systems security.

I am sure there are other patterns that I missed. I encourage you to think about how an idea of a paper came about, talk to the authors whenever you get a chance, and follow your favorite researchers and look for patterns in their papers (sometimes a series of papers that have clear connections). I am sure you will be able to discover patterns of your own at some point!

Tip 3: Develop a good habit of thinking about research ideas

Sounds good so far? The problem is if you don't practice it, it will just be an empty talk. And if you don't have a good habit, you will most likely forget to practice it. Here are a few tips that can help you towards developing that good habit:

- Realize that the execution of a project is fundamentally different from the generation and formulation of an idea. Do not be completely absorbed by your ongoing project and shut yourself down from the outside world (many students make this mistake). Keep reading some papers periodically, e.g., when a new batch of papers comes out from a conference. The least you can do is to scan through all the paper titles. Read the abstract of a few papers that sound interesting. Then read more carefully if they are *actually* interesting to you (this goes back to your research taste). Make sure you pay attention to research ideas in these papers (not just the technical side of things).
- Take paper reviews seriously. Advisors often offload paper reviews to students and then discuss these papers with students. This is a great chance to learn "how the sausage is made". Normally you will read only published papers with good novelty and impact (also well written). During paper reviews though, you get to see rejected papers and learn why they are considered below the bar.
- Be curious and keep an open mind on what types of papers you might be interested in. My recommendation is "read broadly". In practical fields of computer science (e.g., systems, networking, security, software engineering), we are at a point where each individual field has been more and more mature to the point where many good ideas come from crossovers of different fields. Even though my advisor worked mostly on networking back then, I kept reading papers in systems and program analysis, which has helped me tremendously when I become a professor.
- Attend reading groups and talks often and ask questions. If it is a discussion-heavy meeting such as those organized by your own research group, definitely try to participate. If you feel intimidated to do so (because of lack of knowledge or experience), just know that your advisor is trying to help everyone in the group (especially the junior students), and no one will laugh at you for asking

a “dumb” question or anything like that. If you honestly don’t have anything to ask or contribute to a discussion, try to read the paper to-be-discussed ahead of time to gain an advantage. In fact, you can certainly have a discussion on the things we discussed in this article, e.g., which pattern of thinking allowed the idea of a paper to be formed. Once you start doing this once or twice and feeling rewarded (being able to have a good discussion), you will become more and more comfortable over time. One thing that I really like is having a debate on whether an idea is good (even if it is already published in a good conference). Some students play an attacker role looking for weaknesses and limitations, and the others try to defend the paper. This will give you a sense of how “defensible” an idea is, mimicking the paper review process.

- Informally, talk to your labmates often, get to know them personally, create a nice connection. You will probably spend much more time talking to your peers than to your advisor. So why not make some effort to create a great experience? It feels so nice to be able to start a casual conversation on a research topic, a little debate about a paper, and bounce your ideas off others. One way to set up a conducive environment is to periodically talk to them and ask them about how their project is going and volunteer your genuine feedback. They will likely reciprocate and that can get things going. Other than this, it can be really helpful in situations where you can be stuck for a while on some roadblocks. Talking to others can give you a fresh perspective helping you out (I have certainly benefited from it myself).

Acknowledgment: Thanks to my students for their valuable feedback on an earlier draft of the article.