P1

Interviewer: Could you briefly describe your current practice for code auditing, preferably using a relatively detailed but simple example to describe how you conduct auditing, your purpose, and demonstrate how you identify a vulnerability within it.

P1: Okay, let me give an example. For instance, if there's a smart contract project, I usually check the official website and go to GitHub to obtain some basic requirements for auditing the smart contract. This is the first step, to gather background information or understand the project, usually from its whitepaper.

Interviewer: Is this project, for example, a public open source project or an internal project of your own company?

P1: For open source projects, after obtaining the project, I will probably read through some of its written content, including the white paper and its GitHub README, as shown here. That is to say, for some basic aspects of its design, during the reading process, I will not read very carefully. I only need to know what each specific contract is for. For example, as shown and described here, this.sol file refers to a smart contract file, and what it is used for. This is the first step of reading through.

In the second step, after obtaining the specific code, we will choose to conduct an audit. Usually, we will also perform the audit based on the results of a general review. When we learn that it may be a lending project, or an investment project or a collateral project such as a deposit project, we usually prioritize the collateral logic within it. For example, it may involve a contract called options and futures, so I will look into it.

Interviewer: Let me interrupt. Just now you mentioned several terms, such as investment pledge, and other terms that you usually use in the small field of smart contracts.

P1: Right, yes, usually we choose which project to audit based on these specific proper nouns.

Interviewer: For example, when you need to teach someone else how to conduct code auditing, you would tell them what the standard steps are. If these standard steps are clearly documented and will become a standard workflow within your company, are they something you've come up with on your own, or are they used by all companies?

P1: Not entirely. This was written by myself, but other companies may have differences, but basically it's not too far off. This is an audit workflow. For example, the read me or white paper mentioned earlier is a written description of what this project is about. After obtaining it, I will then proceed to conduct an audit, specifically a general audit. First, I will roughly identify the overall logic. Based on the proposal it mentions, this logic is the main logic, so I will first look at its specific implementation.

So basically the first step is to read through the function names of the entire contract to see what functions the contract as a whole contains. For example, here I can clearly identify that the contract includes functions such as balance off, updating up, and a series of other

functions. Then I will extract from these functions some that I can immediately tell what they do. For example, balance off clearly is used to obtain the corresponding balance. And functions like get founding rate and update founding rate clearly are used to set some key variables. So we can roughly classify them into several types. One type is functions for basic functions that we are relatively familiar with, such as balance off, or update, or set, or get. These basic function functions are only used to set or read some key variables in the contract and are used for some centralized adjustments. This is the first type.

For the second category, we may specifically examine the actual implementation of business logic, that is, the functions of those attributes that users can call. These functions are roughly divided into read and write functions. Reading is accessible to all, but writing is only allowed for administrators. For example, there is an "only owner" identifier here, which means that this function can only be written and manipulated by administrators. This is the first category. The second category is like "liquid", which is called by users. What we are most concerned about are also those functions that can be called by users. This is roughly our second step. After we have determined which contract to audit and how to audit it, our second step is to select the functions to be audited and split their functions from a certain contract. This is the second step.

Interviewer: Previously, you mentioned a VSCode plugin called Solidity. Does it provide the functionality to highlight function names and variable names?

P1: Since this contract is relatively short, if it were longer, we would use Solidity Visual Developer to make a rough judgment. How to judge?[Screen Sharing] You can understand these options as an overview or report. You can see that, in fact, it automatically generates a series of information about the function's scope and role based on the content of these functions. Of course, its format may not be ideal; it should be in Markdown format, read like Markdown, and it will show which parts can be read. I'll save it for you to take a look. This includes the function name, visibility, modifier, and whether it simulates pronunciation. This is a quick overview of the function I need to audit for the entire contract. It also includes some visibility settings, for example, External means it can only be called externally, and internal means it can only be called internally.

Interviewer: After you've looked at this variable and understood what it generally does, how do you then go about discovering its vulnerability?

P1: The first way is that we first try to understand what this function is actually doing. Understanding what this function is actually doing, in this step, in the past when there was no ChatGPT, we would read it manually. For example, I would read it like this: first, I know it defines two things; second, I know it returns a series of data. Where does this data come from? It comes from request liquidation. I roughly know that the name "liquidate" generally means liquidation. So, in general, liquidation means that the user first has to request liquidation. According to its logic, that means we can first request liquidation to know what specific contents liquidation includes, or what things the content of liquidation will change, which are actually things like paper change and credit change. After I know it has requested this data, I continue to read the following series of logic, and I won't go into the specific logic. So, if I find there might be a problem here, how did I find it? When I read to this point, I will carefully consider whether this place is complete, that is, whether it has been fully considered. To put it simply, the most critical thing about a vulnerability is whether it has

been fully considered. For example, if equator paper change is less than 0, can it directly execute the following operations? Do we still need to verify some other things?This step is actually about identifying the most critical point in finding vulnerabilities, which means that you need to understand the entire logic before determining whether it verifies what it should during the liquidation process. If it fails to verify, then it may be a vulnerability. What kind of vulnerability do we usually classify this as? Generally, it is a vulnerability of inconsistency or "inconsistency". This is also a type of vulnerability I recently compiled. It should do something but fails to do it, it should update but fails to update, and it should check but fails to check. If it updates one part but fails to update another part that also needs to be updated synchronously, then we consider this an inconsistent vulnerability. This is one type.

Another approach is that based on the existing domain knowledge I currently have, I will match it to our domain knowledge. For example, what are some of the domain knowledge items in the context of liquidation? For instance, during the liquidation process, insufficient verification of the input amount may occur. When I initiate a liquidation to redeem my funds, a common form of vulnerability is that it may not verify my income amount, and simply liquidate and transfer the funds directly, which is definitely not acceptable. I will search within this framework according to a pattern to check for any corresponding potential vulnerabilities. This could be the second possible method.

Interviewer: Right, it means that in the first approach, you're essentially doing a bottom-up analysis without having the original code, and then trying to understand what still needs to be done but hasn't been. And in the second approach, you actually already know some common mistakes or vulnerabilities from your past experience, and then you match them to see where they are.

P1: Yes, you can look at this column [Screen Sharing], and you can understand that this is my Knowledge Base. This is my Knowledge Base. Of course, the reason I've summarized the Knowledge Base now is that I can find vulnerabilities based on this pattern and code. For example, ChatGPT can also do this, but what we're currently working on is still this project, a project where GPT automatically discovers vulnerabilities.

Interviewer: For those without experience, like some others, is it impossible for them to quickly match through this, and they can only be more inclined to use the first method?

P1: It depends on the individual. One thing is that the more you review, the better. The second is that if there are such things, it also has a name called a check list. You can understand it as a tick list that might be used for the normal operation of machines in a factory. You should tick and check this check list before going live.

Interviewer: Let me ask again. Regarding your Knowledge Base, as you can see, it's listed from top to bottom, and it seems to lack a structure. When you perform matching, you can only rely on your own memory and experience to do so.

P1: The matching process is roughly like this, that is, there are two ways to match. One is to match based on the code content. So, just now this was only a type list. In fact, in this library, since this library might be quite large, we will match based on the content code of a function, the data function code. For example, if we input this function code and then convert it into a functionality through GPT, this refers to the functional description of a function. For instance, let me give an example. Suppose we randomly select one, and it is a description of

a function. Then we match the functional description I input with the functional descriptions in this list, and choose the corresponding vulnerability of the one that is most similar, because what are these data? These are vulnerability data. Each function has a possible vulnerability below it. So, after the matching, then ChatGPT, and generally when I add this, I usually use the key concept to ask questions. This is the functional description, this is the vulnerability description, the vulnerability description of the key concept, and I will ask questions in this way. As for the second category or this, this is the corresponding classification point I mentioned just now. What is it for? This is what you just said, it is to draw inferences from one instance. I will probably do it like this. What if it cannot be found? I will look for its parent class.

Interviewer: Is it based on content? For example, is the key concept a simplification of functionality?

P1: You can understand it this way: functionality is the scenario in which I use this knowledge. Key concept is this piece of knowledge, and we humans have defined a basic human behavior here, which is the behavior of using this knowledge in this scenario.

P1: Right, that is to say, first I will match through functionality. I will construct all these functionalities, about 960 to 1000 of them, into a database. Then, by inputting a functionality, I will match the most similar functionality, find the corresponding key concept, and then ask ChatGPT. That is to say, what I actually match in the end is a key concept. That is, input a piece of code, and then get the most suitable key concept for it. Finally, I will ask questions about the key concept and the code, asking whether it is possible for such a vulnerability to occur in this code.

Interviewer: Essentially, your functionality is to describe what that code does, and then your key concept could be the potential vulnerabilities that might arise within the functionality.

P1: Right, if the key concept is not found, I will look for its category. It belongs to the category of repeated array elements, and then I will ask other key concepts under this category. Can you understand what I mean? First, look for the parent class, then the child class. For example, I think there might be some pricing issues here, that is, there might be pricing problems in between. However, when humans think, human thinking might have some problems, but it seems not to occur, unless it is generalized or extended to something similar. Is there a possibility that there might be some similar problems, but not exactly the same as this one?

Interviewer: The Excel table you just showed is your second method, which uses the Knowledge Base for matching.

P1: Regarding the method of matching the Knowledge Base, yes, actually the first one is also quite similar. In fact, you can find that it also mentions "inconsist" here. The first one can actually be classified into this category. To put it simply, the current general way for the people I've defined, the auditors, to detect vulnerabilities is actually based on the knowledge in their own minds. Is there really a situation where this knowledge emerges from scratch, completely without any initial knowledge to refer to? I haven't done this in our current program yet, but it's indeed inevitable that we'll encounter this requirement. Then this requirement will need GPT to fully simulate human thinking.

Interviewer: What does "starting from 0" mean?

P1: You can see that a very necessary condition for me to detect vulnerabilities just now is that I have so much data, so much functionality, and key concepts to ask about, right? What if I don't have them? If I have nothing to ask, does it mean that vulnerabilities can't be detected? What if you're a newbie auditor, a newbie who knows nothing, and your so-called second category, functionality, and key concepts are empty?

Interviewer: It's equivalent to how you transfer your knowledge to a newbie, for example, how to let a newbie use the Knowledge Base of an export.

P1: Right, so this kind of requirement is relatively difficult. I haven't come up with a good solution yet. The key point is to assume that GPT has absolutely no knowledge of these things and let it find these loopholes from scratch. There are roughly two methods here. One is to feed the results of ChatGPT back to ChatGPT. For example, I first ask what this is for, and then ask, based on this judgment, where do you think the loopholes might occur? For an auditor, they might think the loopholes could occur in the if statement. So what do you think might be missing from the judgment? I would guide it step by step to think like a human. Do you understand what I mean? Ok, this is the first solution.

Then, for the second approach, I won't go through the so-called step-by-step questioning as before. I don't want to guide you anymore. Instead, I will fully rely on ChatGPT's own thinking mode. I will deceive it by saying that there is a vulnerability in this code, and asking it to help me find it. In fact, ChatGPT is more receptive to task-driven prompts, and sometimes it seems not very receptive to question-prompted prompts. For example, Your task is to pinpoint [vulnerabilities]. The second paragraph, we have already confirmed that the contract contains only one exploitable logic bug, is equivalent to me deceiving it and myself. I don't know if there is a vulnerability in this code, but I can ask it while telling it there is one and asking it to help me find it. This approach works very well for ChatGPT. Then, the following steps are all about restricting its output. So, overall, for the current ChatGPT, I have roughly three methods to make it find vulnerabilities: the first is based on checklists, the second is based on functionality, and the third is based on key concepts.

Interviewer: But this is equivalent to you having already selected one of them and then letting it go. Have you ever tried directly uploading this document of yours to ChatGPT?

P1: The effect is very poor, and ChatGPT's output ability is very weak. This is roughly such a comparison. Currently, our third functionality uses this, where contract a, b cannot C. In fact, this contract a is exactly the so-called functionality just mentioned, and B cannot BC is the key concept here. Based on knowledge, what vulnerability? Then this is the second solution, and the last solution just mentioned is the so-called deception, and the third is actually a combination of the two methods, which can also produce output. So, how about their accuracy and output quality? So, what is accuracy? Is there actually any false alarm? And output quality refers to whether the output is a real vulnerability or just a security recommendation. That's why I choose this one instead of the first one, because the first one has too many so-called security recommendations, too many security recommendations, many of which are completely useless.

Interviewer: Let me recap. These three steps, planning, reasoning, and validating, basically constitute the three main steps when you conduct an audit. As for the previous steps, planning and reasoning, they should have been covered just now, and now only the validating stage is left.

P1: When it comes to validating, there's one more thing missing. How do you say "validating"? You may actually notice that whether it's planning or reasoning, the most commonly used methods we have now are actually based on some Knowledge Base. Why do I say we must audit the so-called liquid function instead of auditing the balance off function above? Actually, it's also based on a piece of knowledge. This knowledge is that when, for example, only the owner appears, or in this scenario, I prefer to choose the business function that the user can call for liquidation for auditing, rather than using this audit. Well, this knowledge is also what I'm currently working on, which means it can also be transformed into the same data structure as the checklist just mentioned.

Interviewer: Just like what the plugin generated just now, right?

P1: Those are just for auxiliary purposes; in reality, one still has to acquire genuine knowledge, because the plugin only provides you with a list. What exactly you need to review still depends on your knowledge. So why do I mention the planning list? It's because during the validating process, in my understanding, there is also a so-called list and a Knowledge Base. The validating list consists of different vulnerabilities, and it involves finding ways to identify false positives. In simple terms, the purpose of validating is to reduce false positives. The way to reduce false positives is to verify them. There are roughly several methods in this regard, such as the first checklist approach, the so-called forced questioning or deception approach, and the third is the thinking chain approach. Currently, validating is also divided into several approaches, and the main one now is the symbolic execution approach, which involves automation. Based on the description of the vulnerability returned by reasoning, an exploit script for the vulnerability is automatically generated.

Interviewer: Automatically generate vulnerability exploitation scripts.

P1: The first approach is symbolic execution, which is roughly something like this. The second approach is a manual one, also using the so-called validating list to verify the corresponding vulnerability by asking questions to ChatGPT. That is, you can only ask it what you think the key to this vulnerability is, which variables have not been checked, and ChatGPT will reply with some information. Next, I will provide the entire contract to ChatGPT for you to check whether the variables in this contract violate the corresponding checks and restrictions.

Interviewer: Right, I understand, so it's equivalent to saying that ChatGPT can participate in all three stages of your planning, reasoning, and validating. When using this conversation-based approach to execute your process, what do you think is the biggest challenge you've encountered?

P1: There are quite a few. The first issue is that ChatGPT has insufficient context, resulting in its scope being incomplete, while humans can consider things from a global perspective. For example, if using a checklist, it often produces some false positives. The cause of these false positives is actually because I am querying and detecting at the function level, so sometimes it will produce some false positives. That is, because there are already

checks, corresponding judgments, and corresponding restrictions elsewhere, it still thinks there are vulnerabilities, which is definitely a false positive. In fact, it is still a context issue.

Interviewer: This false alarm means either it reports an error or it underreports.

P1: Right, yes, of course, it usually results in false reporting. Underreporting means omission, and in my opinion, when there is omission, if we check the checklist, I feel that the Knowledge Base is not large enough.

Interviewer: Since you're doing this, there must be several goals. One goal is to accurately identify vulnerabilities, and then you need to report the identified ones to, I can understand it as the owner of the project itself. Another thing is that if you can't find all of them, it actually has no impact on you, right? The more you find, the better, but it's okay if you don't find them all.

P1: Failure to identify all potential issues may have an impact on reputation. Yes, that's roughly it. The number of logical vulnerabilities, or such vulnerabilities in general, is unlimited, meaning it will never be completely secure, and there will always be some undetected problems. Of course, it's not the case for all code; the simpler the code, the fewer such issues there are.

Interviewer: Is there any other challenge in the second stage?

P1: Specifically, I don't remember very clearly either. There might be redundant processes in knowledge. During the planning phase, knowledge might have some redundant processes. Specifically, there are some protected functions that ChatGPT also scanned, which is actually the so-called "only owner" part mentioned earlier. That is, ChatGPT also scanned some functions that didn't need to be scanned, but I've resolved this issue. Some redundant functions that didn't need to be scanned were also scanned by me. This is one aspect. I didn't give it a long enough chain of thought. I simply told it to scan all functions, and it didn't think carefully about which ones to scan and which ones not to. In fact, even when written in great detail, humans may not be able to quickly understand and make judgments.

Interviewer: Even if ChatGPT returns a relatively detailed description, people may not be able to tell that it is wrong.

P1: Because it's too complex, some vulnerabilities are overly complicated, and ChatGPT's descriptions aren't particularly good either. So sometimes I'll add a sentence, for example, I'll say "Please explain it to me line by line according to the specific code." I usually add this sentence because I simply can't understand those things; they're too obscure. Then a problem arises during the reasoning process: the ability to draw inferences from one instance is not strong enough. I might only ask about the key concept of a certain functionality project and not ask about anything else, and then I assume that similar vulnerabilities won't occur.

Another example is that there are some so - called key concepts that simply should not be used in this scenario. At this point, if you force a search, ChatGPT may exhibit some hallucinations, resulting in false positives. This is because ChatGPT sometimes tries to answer your questions forcefully and will start to go off on a tangent.

Another thing is that human thinking is relatively jumpy, with meaningful leaps. So in machines, it's like this. For example, many so-called math-oriented knowledge related to mathematics, but without very special logical errors, is evaluated, so basically it's all in the realm of uncertainty. Then I need to continue to improve the vulnerability library, increase knowledge, and expand knowledge. The more general knowledge there is, the more effective it will be after coordinating with planning, making it more capable of thinking leaps like humans. To put it simply, for example, if I match 10 pieces of knowledge, ChatGPT will select the first 5, and the last 5 may be missed. How can I make ChatGPT also scan out the last 5? To put it simply, I remove the first 5, extract them from the ranking, and move the last 5 forward. This is the general idea of the solution. I don't know if you can understand what I'm saying. If the first 5 are all of one category, I may group them together or take a closer look at them. I try to make the so-called key concepts it matches as diverse as possible, which is my ultimate goal. The better its diversity, the closer it can get to the result of human jumpy thinking.

Interviewer: So your goal is to optimize ChatGPT, improve your usage methods, reduce false alarm rates, and increase efficiency.

P1: Right, reducing false positive rates and increasing effectiveness are my current goals. Additionally, humans can actually understand most of the results returned by ChatGPT, but this is after all manual work, not automated. I think automated validating might still be a matter of context, because when humans write so-called attack scripts, they actually comprehensively consider the front and back limitations or code of the entire project, but programs actually find it difficult to consider such comprehensive code or some limitations. Because an attack script itself can be understood as a sub-function, but this sub-function calls almost all other functions of the project, which imposes a requirement that you need to have an understanding of all other functions, and humans can achieve this.

Interviewer: So basically ChatGPT can't write this attack script now.

P1: Let me put it this way. It has two indicators. One is grammatical correctness, and in terms of grammatical correctness, it's okay, but in terms of logical comprehensiveness, it's not okay. Right, that's about it.

Interviewer: The question just asked used the term "challenge". Could you demonstrate how you interact with ChatGPT?

P1: One of the things I did before was the first solution, the manual solution, which involves inputting this thing and then directly copying the long prompt I just mentioned to it, because I've been adjusting it for a long time. Then at this point, ChatGPT will give an answer that looks very reasonable, seemingly very much like it, but sometimes it's not.

Interviewer: So it's equivalent to starting a conversation from scratch every time.

P1: It can be understood in this way. Each function has a possible occurrence of a new vulnerability each time.

Interviewer: So there are many repetitive steps in it.

P1: Right, to put it bluntly, this prompt, look at where the prompt I just found came from, I dug it out from the chat history. In fact, I don't have a good place to store this prompt. Ok, look at this part, it can give a result that looks very reasonable, and it does it very reliably,

but there may be two situations. First, it's too complex, what it writes is quite complex. So at this time, it will be asked to explain this vulnerability in detail, step by step with reference to the code, and basically finish the output, and then ask it to output this.

Interviewer: Do you usually wait until it has fully generated before looking at its response?

P1: Not entirely. For example, it provides so many complex things here. I will focus on the specific content of some of the code it provides. For instance, one of the most critical aspects is that it is related to the domain knowledge of my smart contract, and here there is a term called "negative number". In the domain knowledge of smart contracts, the term "negative number" rarely appears because smart contracts can only handle positive numbers. So, once a negative number appears, I need to pay attention. Sometimes ChatGPT will produce something where it doesn't seem to know that negative numbers can exist in smart contracts. That's why, as you can see, I specifically mentioned in the prompt that all numbers in the code are positive, all are positive, all numbers are positive. But apparently, GPT doesn't seem to have fully considered this. So I need to look back and see if the so-called liquid paper change could be a negative number. Looking back at the original code, it is int256, not uint256. The so-called positive number actually means there must be a "u" in front, uint256. Here it is int256, indicating that its definition itself can be negative.

Ok, I roughly understand that it's possible for this to be a negative number, so ok, ChatGPT doesn't seem to be wrong here. It says there's indeed no problem, and it could indeed be a complex number. Then, after it becomes liquid paper, it only has some calculations, and perhaps it's judged to be a vulnerability or something. I'll still look into it. Another point is that if I don't want to think too much, I'll simply tell it that I don't think it's a vulnerability and ask it to take a look. To put it bluntly, I'm just following the deceptive train of thought in the prompt just now, getting it to refute my opinion. If it can refute successfully, it might indicate that it's probably a vulnerability; if it agrees with my view, it might mean it's not a vulnerability.

In my opinion, ChatGPT does take into account opposing views in this regard. You can see that in this section, it acknowledges its own error in the analysis, but upon reexamining the code, it notes that it has already considered the negative number scenario. So, we might look back at this part above to see where it was considered. Just now, what I read was this section, where it should consider the situation if it is less than 0. Its conclusion is okay. If we put it this way, I would usually skip this loophole because if I go through it quickly, but if I take my time, I will carefully read to see if it could potentially be a problem.

Interviewer: So in this case, it is more likely that there is a problem with its judgment.

P1: I feel that it's probably around a 73% proportion. If I reply with this statement and it says it's not a vulnerability, I feel that there's a 70% chance it should not be one, but there's a 30% possibility that it might still be a vulnerability. It's just that I haven't looked into it carefully, so I think I should continue to look into those.

Interviewer: This time, your interaction with ChatGPT, such as your debate with it, that is, you refuting its viewpoints. For example, can these interactions be repurposed for other code when you interact with it next time?

P1: Right, there's another type that might not be particularly easy to repurpose into other prompts. For example, after I just read this, it mentions negative numbers, but it seems that negative numbers have already been considered here. At this point, I would ask about it, but it seems that negative numbers less than 0 have already been considered. How to explain this? I would still ask it like this. Sometimes in such cases, it actually admits its mistakes, but sometimes it also refutes me. Look at its second sentence, yet the potential issue may no longer be whether it considers handling negative numbers.

I get really annoyed when there are so many, and actually sometimes what bothers me the most is precisely these things, that is, I have to verify whether it is actually a vulnerability. This is actually a rather painful aspect for me, because in my view, this prompt, after it has been executed a sufficient number of times, is capable of finding all the vulnerabilities within this function. That is to say, it has an advantage, which I feel is beneficial, that is, this prompt can be somewhat convergent or the vulnerability can be slightly convergent. ChatGPT's vulnerability in answering questions can be convergent. So sometimes I basically keep going back and forth to conduct secondary questioning. In fact, in the first few times of secondary questioning, it will probably output some different vulnerabilities, but in the end, it will increasingly resemble or be exactly the same as the previous vulnerabilities.

Right, so sometimes after I generally find this thing, I won't ask about it again. Because for the person submitting the vulnerability, it falls into two scenarios. One scenario is the internal audit of the project company, which must ensure that the report you provide to the project party is correct, without false positives. However, bug bounty or vulnerability bounty, or audit competition, usually doesn't consider whether the vulnerability you provided has false positives. There is someone on that side to help you check. There is a specific position, an identity called warden, which is used to check whether the information of the vulnerabilities provided by all participants is a real vulnerability. There will be such a check, and this check actually saves us effort. That is to say, after I generate 100 vulnerabilities, for example, after completing this task, I will write the vulnerability into an English description, including English translation, including description, recommendation, title, and maybe add a severity. Generally, we are required to submit these 4 items, and it will generate a relatively complete vulnerability report. At this time, I can directly copy and paste it onto their platform.

P1: Naturally, someone will check. I don't want to deal with this, so this is also a place where we've always wanted to take a bit of a shortcut. Well, there might be another aspect here, that is, sometimes we'll modify this part, because there's also an unwritten rule here. If you describe a vulnerability using method a, it might be a low-dimensional one, but if you describe it using method b, it might be a high-dimensional one. This has a significant impact on us, especially on our bounty, so I'll add a sentence later to describe this vulnerability as extremely serious. Right, there are indeed actual cases, there really are, but not many, there are truly actual cases. It will give a different description, a very serious one. Right, there are probably some shortcuts taken in there. I can give an example, like some of the audits I did before. You can take a direct look, you should be able to see this too. This is a report on the number of vulnerabilities I submitted before.

For example, after clicking on this, there is a "my submission", where you can see that in this submission, I submitted 3 critical issues, 1 media issue, and a total of 9 issues. However, only 5 were approved, identified, and confirmed as vulnerabilities, which means 4

were not vulnerabilities and would be rejected by Secure Three. You can see why they were rejected. For example, "Merge the same issue for out of scope" or "Please give us more detail about the specific danger also".

P1: Right, this is the result of an audit I gave them, and I really haven't read the feedback they gave me. It seems they probably didn't understand it; what I wrote was too abstract, because this was also generated by GPT.

Interviewer: Is its acceptance rate high later?

P1: All these 8 vulnerabilities were generated by ChatGPT, and all were found by ChatGPT. Moreover, it has a scoring system or evaluates whether your submission is well-written. Because I've talked to some auditors, and they said they think the audit of this code is quite good. They think this waiting is, so to speak, a good finding, a good submission. Some scores are not very high. Basically, from here on, all were submitted by myself, and all were found by GPT. For example, there are 34 in this, and all 34 were found by GPT, and 21 were provided. This is where I earned the most, about $23,000. Right.

Interviewer: Okay, and now I'd like to ask the last question. If we were to design a new interface to help you interact with ChatGPT, what expectations do you have and what features would you like it to have?

P1: Actually, I understand this part. I don't know if you've looked at the Audit Wizard. For auditors, I think there may be two different levels of requirements. The first, with relatively high expectations, is something like the Audit Wizard, which not only can ask questions about code but also can integrate with some different tools or other things. This is perhaps what we ultimately hope for. Of course, the Audit Wizard itself is not particularly good; it doesn't make much use of AI. In contrast, the other requirement is that we may not import corresponding code but simply copy a contract file in. If that's the case, expectations may be higher. It would be best if there were some very effective trick prompts available, like the prompt I used before. If there were such prompts, I think it would be better. Because, to be honest, the completeness of the prompt greatly affects GPT's response results, which means that the better the prompt, the harder it is to obtain. The prompt I have here is one that I adjusted after a long time. If there were a collection of such prompts, I think it would be better.

Interviewer: Do you think you used other traditional software a lot in the past, or do you prefer to rely on your own checklist?

P1: In fact, we all have such scanning requirements, but such requirements actually have some drawbacks. The reason for saying there are drawbacks now is that there are some things it can't scan out. To put it simply, these things are all static engines or purely program analysis tools, and their drawback is that they are unable to recognize business logic and unable to understand business logic.

Interviewer: How much do you think GPT has improved your efficiency approximately?

P1: For me, it may have improved significantly. I basically don't think at all when auditing now. Previously, it took 100 units of time, and now it might be only 1/10, or about 1/10. To put it simply, just look at the things I asked just now. For example, in the past, after getting this code, I would carefully examine each one, and the time was uncertain. Manually, the

efficiency was about 300 to 500 lines per day, with 8 hours a day. If it's ChatGPT, it might take 30 to 50 minutes at most, and this still includes the time for you to keep asking questions, verifying, and having conversations. If you completely ignore the output effect and just want to get the job done for the money, it can be completed in 3 to 5 minutes. It used to take 8 hours, or perhaps several hundred minutes, around four or five hundred minutes.

Interviewer: How much do you think your performance has improved approximately?

P1: I think there's at least a doubling of performance improvement, because there are some things that I really can't tell if I look at them myself, at least a doubling is possible.

Interviewer: Okay, that should be all the questions.