Secureum Bootcamp Audit Techniques & Tools 101 Quiz

November 21, 2021 / patrickd

This is a writeup of the Secureum Bootcamp Audit Techniques & Tools 101 Quiz containing solutions and references to the provided study material.
For fairness it was published after submissions to it were closed.

The quiz consisted of 16 questions with an overall strict timelimit of 16 minutes. The ordering of the questions was randomized, so the numbering here won't match with the numbering elsewhere.

> "Which of the below is/are accurate?"
>
> — 1 of 16

- [ ] A. Audits identify all security vulnerabilities and guarantee bug-free code
- [ ] B. Audits cover only smart contracts but never the offchain code
- [x] C. Audits suggest fixes for issues identified and aim to reduce risk
- [ ] D. None of the above

▼ Solution

Correct is C.

> Audit Non-goal: Audit is not a security guarantee of "bug-free" code by any stretch of imagination but a best-effort endeavour by trained security experts

> operating within reasonable constraints of time, understanding, expertise and of course, decidability.

from point 4 of Security Audit Techniques & Tools 101 - by Secureum

> Audit Scope: For Ethereum-based smart-contract projects, the scope is typically the on-chain smart contract code and sometimes includes the off-chain components that interact with the smart contracts.

from point 2 of Security Audit Techniques & Tools 101 - by Secureum

> Audit Goal: The goal of audits is to assess project code (with any associated specification, documentation) and alert project team, typically before launch, of potential security-related issues that need to be addressed to improve security posture, decrease attack surface and mitigate risk.

from point 3 of Security Audit Techniques & Tools 101 - by Secureum

> Audit: It detects and describes (in a report) security issues with underlying vulnerabilities, severity/difficulty, potential exploit scenarios and recommended fixes.

from point 1.1 of Security Audit Techniques & Tools 101 - by Secureum

> “Audit reports from audit firms typically include”
>
> — 2 of 16

- [x] A. Finding likelyhod/difficulty, impact and severity
- [x] B. Exploit scenarios and recommended fixes
- [ ] C. Formal verification of all findings with proofs and counterexamples
- [ ] D. All of the above

▼ Solution

Correct is A, B.

> Audit Reports: include details of the scope, goals, effort, timeline, approach, tools/techniques used, findings summary, vulnerability details, vulnerability

> classification, vulnerability severity/difficulty/likelihood, vulnerability exploit scenarios, vulnerability fixes and informational recommendations/suggestions on programming best-practices.

from point 13 of Security Audit Techniques & Tools 101 - by Secureum

> Audit Findings Classification: The vulnerabilities found during the audit are typically classified into different categories which helps to understand the nature of the vulnerability, potential impact/severity, impacted project components/functionality and exploit scenarios.

from point 14 of Security Audit Techniques & Tools 101 - by Secureum

> “These audit techniques are especially well-suited for smart contracts (compared to Web2 programs)”
>
> — 3 of 16

- [x] A. Formal verification because contracts are relatively smaller with specific properties
- [x] B. Fuzzing because anyone can send random inputs to contracts on blockchain
- [x] C. Static source-code analysis because contracts are expected to be open-source
- [x] D. High-coverage testing because contract states and transitions are relatively fewer

▼ Solution

Correct is A, B, C, D.

> Testing: Smart contract testing has a similar motivation but is arguably more complicated despite their relatively smaller sizes (in lines of code) compared to Web2 software

from point 22.1 of Security Audit Techniques & Tools 101 - by Secureum

> Fuzzing is especially relevant to smart contracts because anyone can interact with them on the blockchain with random inputs without necessarily having a

> valid reason or expectation (arbitrary byzantine behaviour)

from point 24.1 of Security Audit Techniques & Tools 101 - by Secureum

> It is natural for this Q to be open to interpretation because it is a 100K ft level question comparing techniques "especially well-suited" to web3 smart contracts vis-a-vis web2 software, which covers a lot of ground. It is not about what is possible or what is expected but about suitability. All these techniques can be and are performed on web2 applications but generalized aspects of size, scope, nature of user-interactions, source-code availability/expectations and reduced states/transitions of smart contracts make the techniques more suitable for them compared to web2 software. In fact, @dguidotalks specifically about these aspects in what web2 can learn from web3 about security in his SafeCast interview here:
> https://twitter.com/0xRajeev/status/1454273518154051586

from Rajeev on Secureum Discord

> “The following kinds of findings may be expected during audits”
>
> — 4 of 16

- [x] A. True positives after confirmation from the project team
- [x] B. False positives due to assumptions from missing specification and threat model
- [x] C. False negatives due to limitations of time and expertise
- [ ] D. None of the above

▼ Solution

Correct is A, B, C.

> Findings may be contested as not being relevant, outside the project’s threat model or simply acknowledged as being within the project’s acceptable risk model

from point 89.2 of Security Audit Techniques & Tools 101 - by Secureum

> False Positives: are findings which indicate the presence of vulnerabilities but which in fact are not vulnerabilities. Such false positives could be due to incorrect assumptions or simplifications in analysis which do not correctly consider all the factors required for the actual presence of vulnerabilities.

from point 28 of Security Audit Techniques & Tools 101 - by Secureum

> False Negatives: are missed findings that should have indicated the presence of vulnerabilities but which are in fact are not reported at all. Such false negatives could be due to incorrect assumptions or inaccuracies in analysis which do not correctly consider the minimum factors required for the actual presence of vulnerabilities.

from point 29 of Security Audit Techniques & Tools 101 - by Secureum

> Auditors generally collate all findings from their review into a report which is handed to the project team. At this point, the assumption from the auditors is that all the findings in their report are true positives. However, depending on the differing threat/trust models or different assumptions made between the audit & project teams, some of the findings may be treated as false positives by the project team which thereafter may choose to ignore such findings, recognize but not act (via fixes) on them, etc. Auditors internally may bring up their individual findings with other team members to discuss if they are indeed true/false. They may also bring up doubtful findings with the project team during an audit (interim discussions/clarifications). Findings which are deemed by everyone as false positives, i.e. irrelevant, will not be included in the report. There may also be disagreements between the auditors & project teams about the threat/trust models, assumptions or difficulty/severity levels, which may lead to opposing viewpoints which are sometimes documented in the reports. But, in general, many of the reported findings are "confirmed" by the projects, after which we can think of them as true positives.

from Rajeev on Secureum Discord

> “Which of the following is/are true?”
>
> — 5 of 16

- [ ] A. Audited projects always have clear/complete specification and documentation of all contract properties
- [x] B. Manual analysis is typically required for detecting application logic vulnerabilities
- [ ] C. Automated tools like Slither and MythX have no false negatives
- [ ] D. The project team always fixes all the findings identified in audits

▼ Solution

Correct is B.

> Very few smart contract projects have detailed specifications at their first audit stage. At best, they have some documentation about what is implemented. Auditors spend a lot of time inferring specification from documentation/implementation which leaves them with less time for vulnerability assessment.

from point 20.3 of Security Audit Techniques & Tools 101 - by Secureum

> Manual analysis is however the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found

from point 27.3 of Security Audit Techniques & Tools 101 - by Secureum

> Automated analysis using tools is cheap (typically open-source free software), fast, deterministic and scalable (varies depending on the tool being semi-/fully-automated) but however is only as good as the properties it is made aware of, which is typically limited to Solidity and EVM related constraints

from point 27.1 of Security Audit Techniques & Tools 101 - by Secureum

> Findings may be contested as not being relevant, outside the project's threat model or simply acknowledged as being within the project's acceptable risk model

from point 89.2 of Security Audit Techniques & Tools 101 - by Secureum

> “Automated tools for smart contract analysis”
>
> — 6 of 16

- [ ] A. Are sufficient therefore making a manual analysis unnecessary
- [ ] B. Have no false positives whatsoever
- [ ] C. Are best-suited for application-level vulnerablities
- [x] D. None of the above

▼ Solution

Correct is D.

> Manual analysis is however the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found

from point 27.3 of Security Audit Techniques & Tools 101 - by Secureum

> Automated analyzers do not understand application-level logic and their constraints. They are limited to constraints/properties of Solidity language, EVM or Ethereum blockchain.

from point 83.1 of Security Audit Techniques & Tools 101 - by Secureum

> Smart contract security tools are useful in assisting auditors while reviewing smart contracts. They automate many of the tasks that can be codified into rules with different levels of coverage, correctness and precision. They are fast, cheap, scalable and deterministic compared to manual analysis. But they are also susceptible to false positives. They are especially well-suited currently to detect common security pitfalls and best-practices at the Solidity and EVM level. With varying degrees of manual assistance, they can also be programmed to check for application-level, business-logic constraints.

from point 79 of Security Audit Techniques & Tools 101 - by Secureum

> “Which of the following is/are true?”
>
> — 7 of 16

- [x] A. Slither supports detectors, printers, tools and custom analyses
- [ ] B. Echidna is a symbolic analyzer tool
- [x] C. MythX is a combination of static analysis, symbolic checking and fuzzing tools
- [ ] D. None of the above

▼ Solution

Correct is A, C.

> Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

from point 33 of Security Audit Techniques & Tools 101 - by Secureum

> Echidna is a Haskell program designed for fuzzing/property-based testing of Ethereum smart contracts.

from point 45 of Security Audit Techniques & Tools 101 - by Secureum

> MythX is a powerful security analysis service that finds Solidity vulnerabilities in your Ethereum smart contract code during your development life cycle. It is a paid API-based service which uses several tools on the backend including a static analyzer (Maru), symbolic analyzer (Mythril) and a greybox fuzzer (Harvey) to implement a total of 46 detectors.

from point 56 of Security Audit Techniques & Tools 101 - by Secureum

> “Which of the following is/are correct about false positives?”
>
> — 8 of 16

- [x] A. They are findings that are not real concerns/vulnerabilities after further review

- [ ] B. They are real vulnerabilities but are falsely claimed by auditors as benign
- [x] C. They are possible with automated tools
- [ ] D. None of the above

▼ Solution

Correct is A, C.

> False positives require further manual analysis on findings to investigate if they are indeed false or true positives

from point 28.1 of Security Audit Techniques & Tools 101 - by Secureum

> Smart contract security tools are useful in assisting auditors while reviewing smart contracts. They automate many of the tasks that can be codified into rules with different levels of coverage, correctness and precision. They are fast, cheap, scalable and deterministic compared to manual analysis. But they are also susceptible to false positives.

from point 79 of Security Audit Techniques & Tools 101 - by Secureum

> “Audit findings”
>
> — 9 of 16

- [x] A. May include both specific vulnerabilities and generic recommendations
- [x] B. May not all be fixed by the project team for reasons of relevancy and acceptable trust/threat model
- [ ] C. Always have demonstratable proof-of-concept exploit code on mainnet
- [ ] D. None of the above

▼ Solution

Correct is A, B.

> It detects and describes (in a report) security issues with underlying vulnerabilities, severity/difficulty, potential exploit scenarios and recommended fixes.

from point 1.1 of Security Audit Techniques & Tools 101 - by Secureum

> It also provides subjective insights into code quality, documentation and testing.

from point 1.2 of Security Audit Techniques & Tools 101 - by Secureum

> Findings may be contested as not being relevant, outside the project's threat model or simply acknowledged as being within the project's acceptable risk model

from point 89.2 of Security Audit Techniques & Tools 101 - by Secureum

> Codified exploits should always be on a testnet, kept private and responsibly disclosed to project teams without any risk of being actually executed on live systems resulting in real loss of funds or access

from point 99.2 of Security Audit Techniques & Tools 101 - by Secureum

> "Which of the following is/are typical manual review approach(es)?"
>
> — 10 of 16

- [x] A. Asset flow
- [ ] B. Symbolic checking
- [x] C. Inferring constraints
- [x] D. Evaluating assumptions

▼ Solution

Correct is A, C, D.

> Audit Techniques: Symbolic checking (automated)

from point 19.6 of Security Audit Techniques & Tools 101 - by Secureum

> Manual review approaches: Auditors have different approaches to manual reviewing smart contract code for vulnerabilities. [...] Starting with access

> control [...] Starting with asset flow [...] Inferring constraints [...] Evaluating assumptions

from point 90 of Security Audit Techniques & Tools 101 - by Secureum

> “Access control analysis is a critical part of manual review for the reason(s) that”
>
> — 11 of 16

- [ ] A. It is the easiest to perform because smart contracts never have access control
- [ ] B. It is the fastest to perform because there are always only two roles: users and admins
- [x] C. It is fundamental to security because privileged roles (of which there may be many) may be misused/compromised
- [ ] D. None of the above

▼ Solution

Correct is C.

> While the overall philosophy might be that smart contracts are permissionless, in reality, they do indeed have different permissions/roles for different actors who interact/use them.

from point 91.1 of Security Audit Techniques & Tools 101 - by Secureum

> Starting with access control: Access control is the most fundamental security primitive which addresses ‘who’ has authorised access to ‘what.’ (In a formal access control model, the ‘who’ refers to subjects, ’what’ refers to objects and an access control matrix indicates the permissions between subjects and objects.)

from point 91 of Security Audit Techniques & Tools 101 - by Secureum

> The general classification is that of users and admin(s). For purposes of guarded launch or otherwise, many smart contracts have an admin role that is typically the address that deployed the contract. Admins typically have control over

> critical configuration and application parameters including (emergency) transfers/withdrawals of contract funds.

from point 91.2 of Security Audit Techniques & Tools 101 - by Secureum

> “Which of the following is/are true about vulnerability difficulty and impact?”
>
> — 12 of 16

- [ ] A. Difficulty indicates how hard it was for auditors to detect the issue
- [ ] B. Difficulty is an objective measure that can always be quantified
- [x] C. Impact is typically classified as High if there is loss/lock of funds
- [ ] D. None of the above

▼ Solution

Correct is C.

> Audit Findings Likelihood/Difficulty: Per OWASP, likelihood or difficulty is a rough measure of how likely or difficult this particular vulnerability is to be uncovered and exploited by an attacker.

from point 15 of Security Audit Techniques & Tools 101 - by Secureum

> Many likelihood and impact evaluations are contentious and debatable between the audit and project teams, typically with security-conscious audit teams pressing for higher likelihood and impact and project teams downplaying the risks.

from point 100.3 of Security Audit Techniques & Tools 101 - by Secureum

> If there is any loss or locking up of funds then the impact is evaluated as High. Exploits that do not affect funds but disrupt the normal functioning of the system are typically evaluated as Medium. Anything else is of Low impact.

from point 100.2 of Security Audit Techniques & Tools 101 - by Secureum

> “Application-level security constraints”
>
> — 13 of 16

- [ ] A. Are always clearly/completely specified and documented
- [x] B. Have to be typically inferred from the code or discussions with project team
- [x] C. Typically require manual analysis
- [ ] D. None of the above

▼ Solution

Correct is B, C.

> Auditors may need to infer business logic and their implied constraints directly from the code or from discussions with the project team and thereafter evaluate if those constraints/properties hold in all parts of the codebase.

from point 83.3 of Security Audit Techniques & Tools 101 - by Secureum

> [...] However, application-level constraints are rules that are implicit to the business logic implemented and may not be explicitly described in the specification e.g. mint an ERC-721 token to the address when it makes a certain deposit of ERC-20 tokens to the smart contract and burn it when it withdraws the earlier deposit. Such constraints may have to be inferred by the auditors while manually analyzing the smart contract code.

from point 95 of Security Audit Techniques & Tools 101 - by Secureum

> “Which of the following is/are typically true?”
>
> — 14 of 16

- [ ] A. Static analysis analyzes program properties by actually executing the program
- [ ] B. Fuzzing uses valid, expected and deterministic inputs
- [ ] C. Symbolic checking enumerates individual states/transitions for efficient state space traversal
- [x] D. None of the above

▼ Solution

Correct is D.

> Static analysis: is a technique of analyzing program properties without actually executing the program.

from point 23 of Security Audit Techniques & Tools 101 - by Secureum

> Fuzzing: or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program.

from point 24 of Security Audit Techniques & Tools 101 - by Secureum

> Instead of enumerating reachable states one at a time, the state space can sometimes be traversed more efficiently by considering large numbers of states at a single step.

from point 25.4 of Security Audit Techniques & Tools 101 - by Secureum

> "Which of the following is/are generally true about asset flow analysis?"
>
> — 15 of 16

- [x] A. Analyzes the flow of Ether or tokens managed by smart contracts
- [x] B. Assets should be withdrawn only by authorized addresses
- [ ] C. The timing aspects of asset withdrawals/deposits is irrelevant
- [ ] D. The type and quantity of asset withdrawals/deposits is irrelevant

▼ Solution

Correct is A, B.

> Starting with asset flow: Assets are Ether or ERC20/ERC721/other tokens managed by smart contracts. Given that exploits target assets of value, it makes sense to start evaluating the flow of assets into/outside/within/across smart contracts and their dependencies.

from point 92 of Security Audit Techniques & Tools 101 - by Secureum

> Who: Assets should be withdrawn/deposited only by authorised/specified addresses as per application logic

from point 92.1 of Security Audit Techniques & Tools 101 - by Secureum

> When: Assets should be withdrawn/deposited only in authorised/specified time windows or under authorised/specified conditions as per application logic (when)

from point 92.2 of Security Audit Techniques & Tools 101 - by Secureum

> What type: Assets, only of authorised/specified types, should be withdrawn/deposited as per application logic

from point 92.6 of Security Audit Techniques & Tools 101 - by Secureum

> How much: Assets, only in authorised/specified amounts, should be withdrawn/deposited as per application logic

from point 92.7 of Security Audit Techniques & Tools 101 - by Secureum

> “Which of the following is/are generally true about control and data flow analyses?”

— 16 of 16

- [x] A. Interprocedural control flow is typically indicated by a call graph
- [x] B. Intraprocedural control flow is dictated by conditionals (if/else), loops (for/while/do/continue/break) and return statements
- [x] C. Interprocedural data flow is evaluated by analyzing the data used as argument values for function parameters at call sites
- [x] D. Interprocedural data flow is evaluated by analyzing the assignment and use of variables/constants along control flow paths within function

▼ Solution

Correct is A, B, C, D.

> Evaluating control flow: Interprocedural (procedure is just another name for a function) control flow is typically indicated by a call graph which shows which functions (callers) call which other functions (callees), across or within smart contracts

from point 93.1 of Security Audit Techniques & Tools 101 - by Secureum

> Evaluating control flow: Intraprocedural (i.e. within a function) control flow is dictated by conditionals (if/else), loops (for/while/do/continue/break) and return statements.

from point 93.2 of Security Audit Techniques & Tools 101 - by Secureum

> Evaluating data flow: Interprocedural data flow is evaluated by analyzing the data (variables/constants) used as argument values for function parameters at call sites

from point 94.1 of Security Audit Techniques & Tools 101 - by Secureum

> Evaluating data flow: Intraprocedural data flow is evaluated by analyzing the assignment and use of (state/memory/calldata) variables/constants along the control flow paths within functions.

from point 94.2 of Security Audit Techniques & Tools 101 - by Secureum

In Blockchain Tags Ethereum, Secureum Bootcamp

← Secureum Bootcamp Audit Findings… Secureum Bootcamp Security Pitfalls …