Announcing the winners of the first Underhanded Solidity Coding Contest

Nick Johnson

Follow

--

3

Listen

Share

After much deliberation, the judgements are in, and we're finally able to announce the three winners of the first Underhanded Solidity Coding Contest. We received 23 qualifying entries in total, and the average quality was very high. Thanks again to everyone who submitted!

All the submissions are now available on Github; note that the README in each contains spoilers provided by the author, so if you want to figure things out for yourself, be sure to read that last.

Without any further ado, the winners:

Honorable Mention: Doug Hoyte

Doug's entry implements a fairly standard token funding contract, with a bug that lets the owner manipulate the contract's storage in unexpected ways.

Feedback from the judges was that while the nature of the exploit was very clever, the bug that enabled it is fairly obvious to an auditor (though easily deniable as a simple mistake), and the 'unused' code looked suspicious. With some refinement of the way the bug was disguised, this could have been a truly devious entry.

We have decided to award a fourth place prize to Doug; for his prize, he will receive 10 MLN tokens.

Doug's writeup in the readme is comprehensive and well worth a read, going into great depth about the subtleties of Solidity and the EVM that enable the exploit.

SPOILERS

The contract features a storage array whose length field can be decremented below 0. This causes an arithmetic overflow, effectively disabling Solidity's array bounds checking. As a result, after the overflow writes to the array can be used to overwrite any storage element located after the array — including all mappings!

Third Place: João Carvalho

João's entry implements a dutch auction crowdsale as a price discovery mechanism for selling tokens. Give the contract a look over if you'd like to figure out the flaws yourself.

For his prize, João received a ticket to DevCon 3. Thanks to the Ethereum Foundation for contributing this prize!

SPOILERS

Although the fallback function captures funds sent to the contract the regular way, the fallback isn't the only way to send funds to a contract. A selfdestruct by another contract can target the fallback, and result in sending funds without executing any code. This can cause the dutch auction to end early, artificially inflating the token price.

Here's what the judges had to say about João's entry:

Yudi

External update to the balance to manipulate price

- Clean, hard to trace back to the team
- Fairly complex

Chris

The submission exploits an often overlooked edge case in the EVM and is thus a good submission per se, but there are much shorter submissions for the same edge case where the problem is equally well “hidden”.

Matthew

very good — i like the selfdestruct transfer technique, though ways that it can be used is restricted and requires somewhat unusual functionality

Second Place: Richard Moore

Richard’s entry implements a standard token contract with a small variation: the owner may only withdraw the raised funds gradually — 1 ether in the first week, 2 in the second, 4 in the third, and so forth. Stop here and give the contract a read yourself if you want to figure out where the deviousness lies.

For his prize, Richard received a flight to Lyon and lunch with the iExec team. Thanks to iExec for sponsoring this prize!

SPOILERS

The increasing schedule of payouts is accomplished by subtracting the remaining balance from the total supply of tokens. Since the fallback function always increases the total supply when it receives ether, this seems fine — only, the fallback function isn’t the only way to fund a contract.

As in João’s entry, it’s possible to send funds to a contract without running its fallback function, by selfdestructing another contract with the token contract as a target address. As demonstrated by Richard’s tests, it’s also possible to prefund the contract before deploying it. Once either of these has happened, the subtraction will cause an overflow, allowing the owner to withdraw as much as they want.

Here’s what the judges have to say about Richard’s entry:

Yudi

External update to the balance to underflow max withdrawal amount

- Short, clean, can be dismissed as a coder oversight
- Overflow/Underflow issue is relatively easy to notice

Chris

Nice exploit of something that is not directly visible in the program, but code could be shorter.

Matthew

cute exploit, fixed by safemath but i like the trigger being funding it before the contract is deployed

First Place: Martin Swende

Martin’s entry implements a ‘round table’ for governance, and raises funds by allowing people to bid on seats around the table. A ‘small honorarium’ is paid out to the creator of the contract for each bid. I highly recommend reading his code and trying to figure out for yourself where the flaw is.

For his prize, Martin agreed to present his solution at DevCon 3 this November.

SPOILERS

The ‘small honorarium’ sometimes turns out to be not quite so small. Martin makes use of the fact that Solidity does not validate that the ABI-encoded length of an array matches the length of the payload. He jumps through a couple of plausible-seeming hoops to avoid referencing the length directly when processing bids — except when he increments the honorarium. As a result, the owner of the contract can submit a specially crafted bid that inflates the amount of the honorarium arbitrarily, and allows them to withdraw all the funds supplied by other users to their own account.

Here’s what the judges had to say about Martin’s entry:

Yudi

Exploiting dynamic array length

- Short, clean, can be dismissed as a coder error, I like the theme :)
- Seems a bit overly complex (no need for a loop), doesn’t actually check msg.value against the bids

Christian

Bonus points for actually taking a look at compiled code.

Suspicious that seats.length == bids.length is checked, but not seats.length == NO_OF_SEATS_BID

Relies on the fact that the ABI decoder does not check for overflows, but still: The actual problem is that bids.length is assumed to equal NO_OF_SEATS_BID but this assumption is not checked anywhere, the overflow only helps in executing the exploit.

Matthew

Impressive understanding of ABI and how solidity handles its own type structures, probably would not have caught the intention in an audit even though i may have caught the bug, also really like the fact that it allows the owner to act in a way where he may not even have to use the bug

Wrapping Up

A huge thanks to everyone who submitted, and especially to our hardworking judges, who had to read and rate over 20 deliberately difficult to audit contracts. Look out for a new USCC, with a new theme, next year!