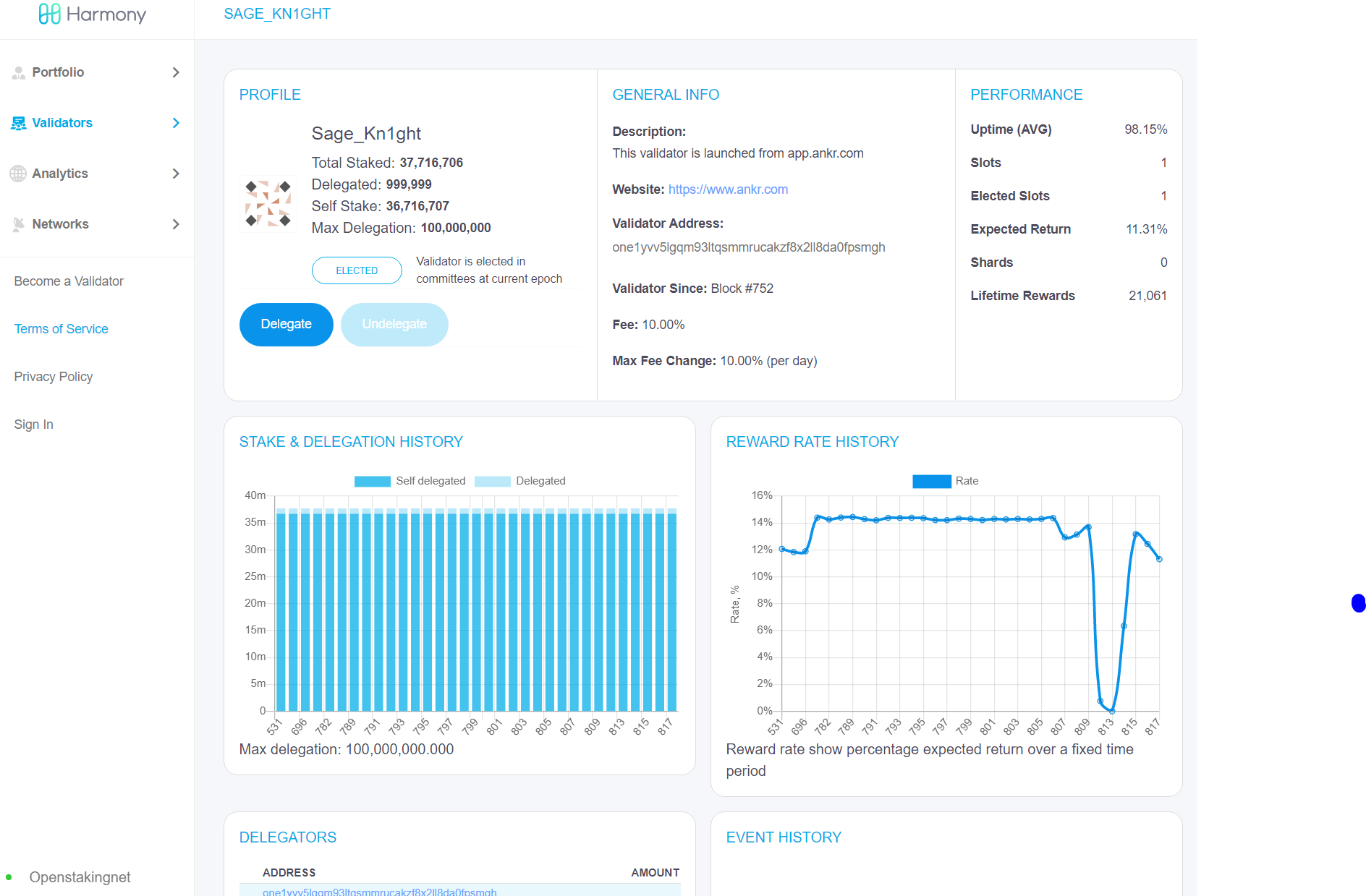
**Harmony Exploit Scenarios**

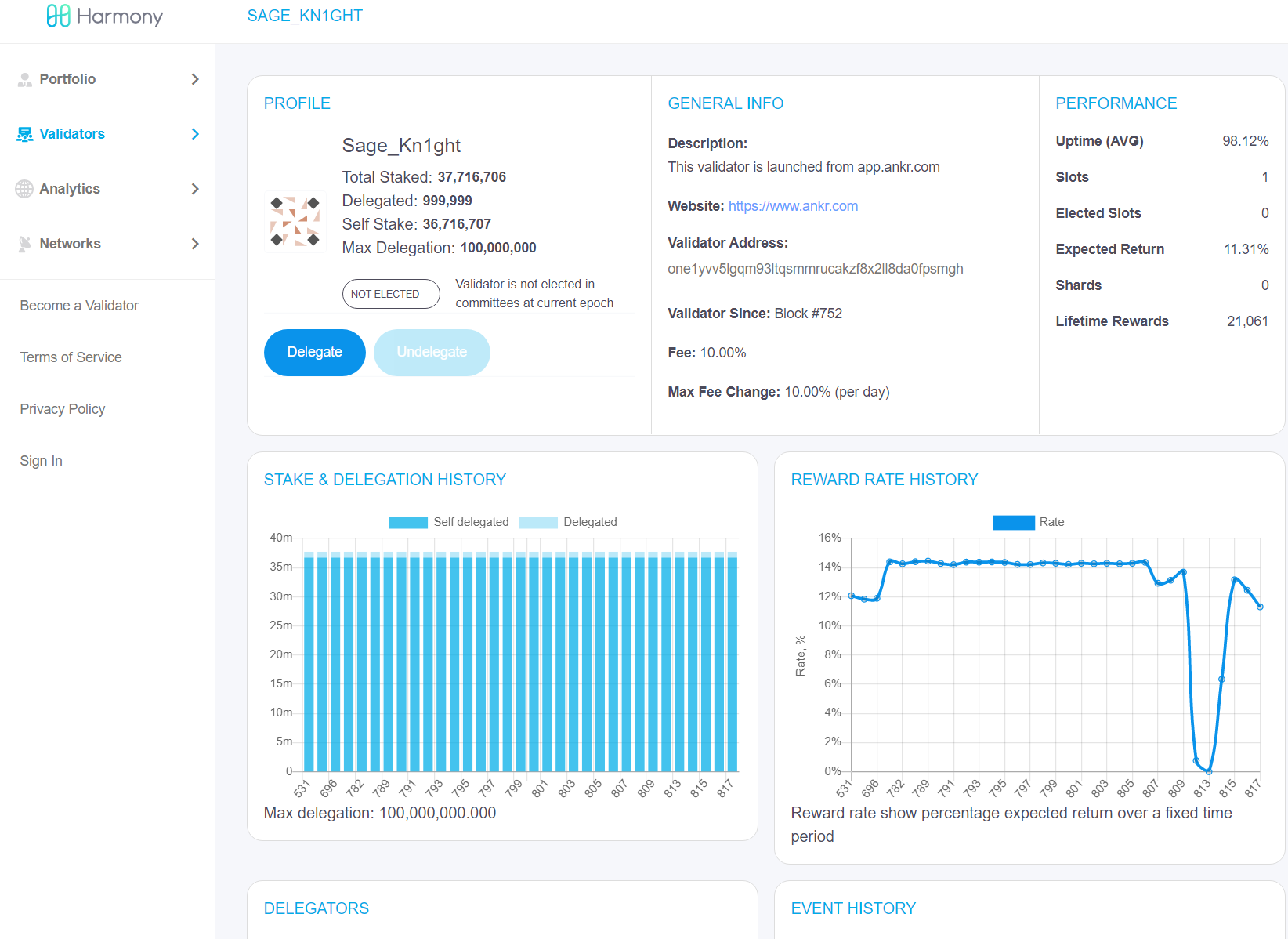
**Scenario 1:** Create a validator with a new ONE address and new BLS key but with the same identity(name) as the one that exists.

**Target:** Sage\_Kn1ght

**Before running the create validator command**



**After running the create validator command**



For scenario 1 there are 3 outcomes:

- stakers can possibly delegate to fake validator identities. This can especially happen at the launch of the network, when the delegated staked amount doesn't differ too much between the validators.

- a validator with a fake identity can behave malicious and generate a bad reputation for the real identity;

- A weird behavior has been noticed for 2 out of 7 targets. For these 2 targets the validator moved from "Elected" to "Not Elected" status, once the fake validator was created. Here maybe it would make sense to create a script which will try to create multiple validators for already exisiting identities. However, if the problem will be mitigated, also because of outcome 1, then maybe it is not worth to invest time in running further tests.

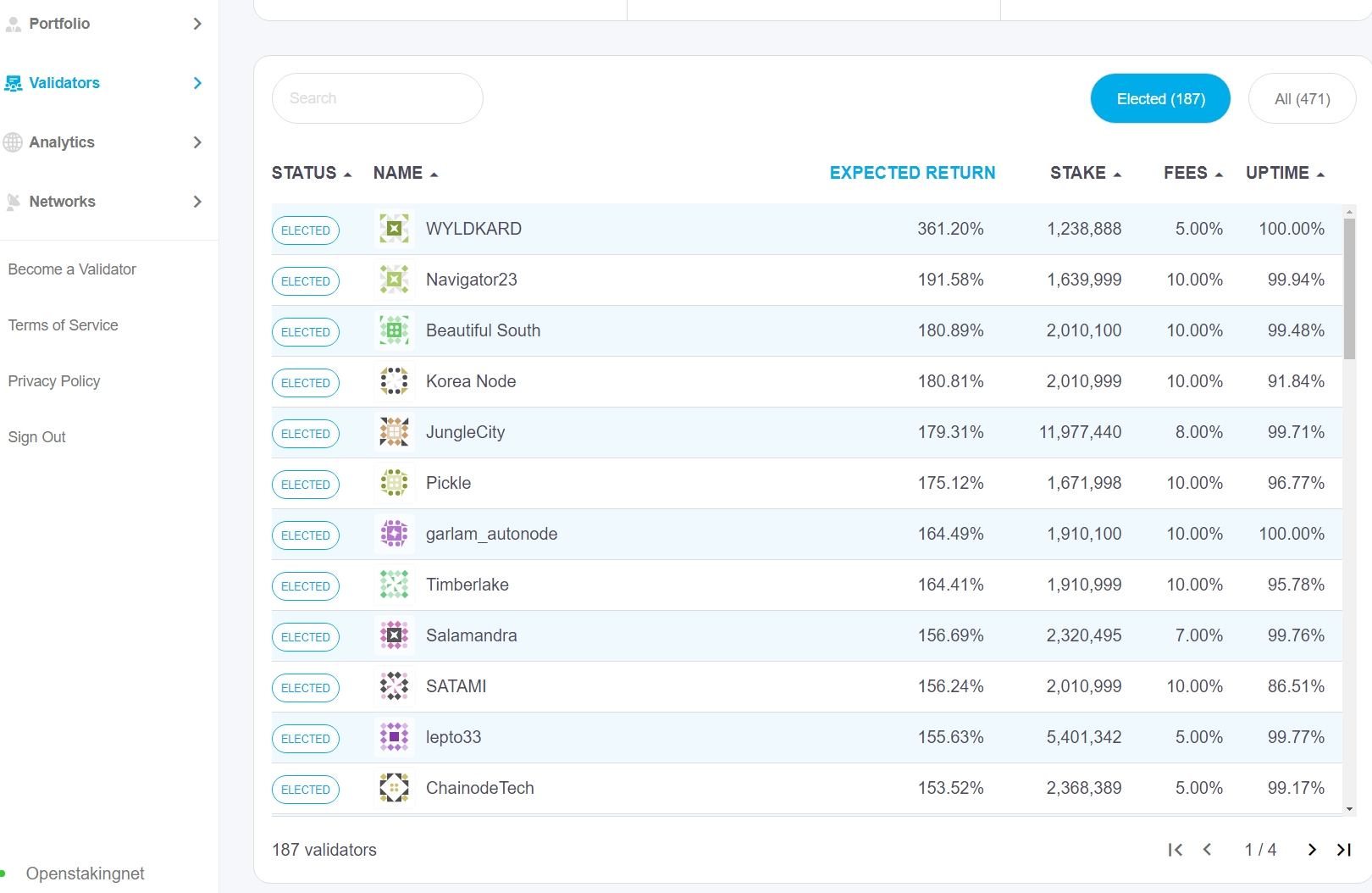
**Possible Mitigation:** keybase integration or other solution for verifying validator identity. The keybase integration would also solve the problem of validator logos, as that can be picked up from keybase. Solving this problem will also keep the validator list clean and unique.

Regarding validator verification using keybase: <https://github.com/solana-labs/solana/issues/5187>

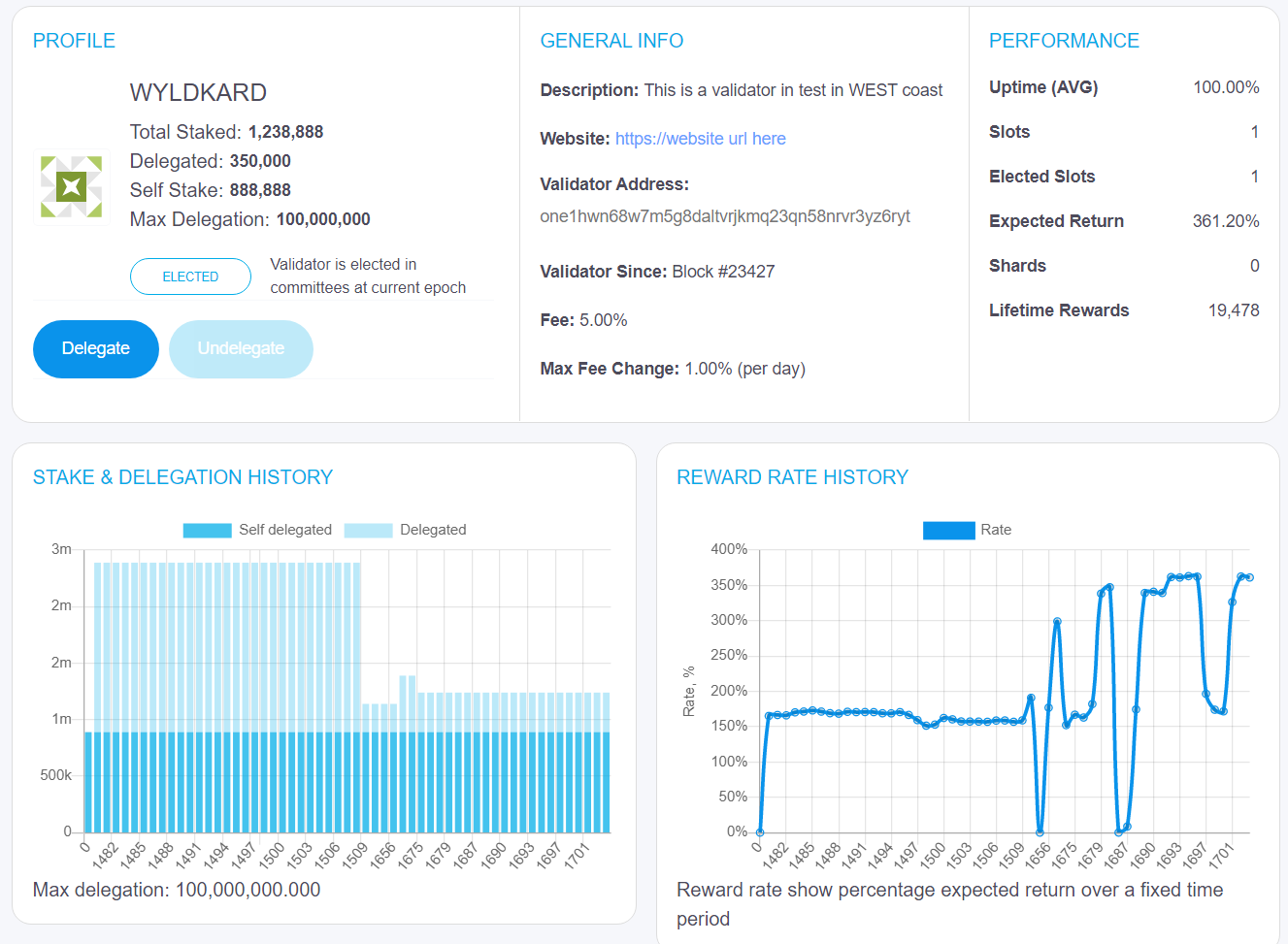
**Possible further tests:** maybe through a script which will try to create multiple validators with existing names. However, if the problem will be mitigated, also because of outcome 1, then maybe it is not worth to invest time in running further tests.

**Scenario 2:** Manipulate the rewards a validator can earn by manipulating the expected return value a validator would offer through repeated delegations/undelegations. By doing this, stakers will have the false impression that the validator offers a poor expected return value and avoid to delegate to this validator.

In this case **WYLDKARD** was selected as target as it was the validator with the highest expected return.

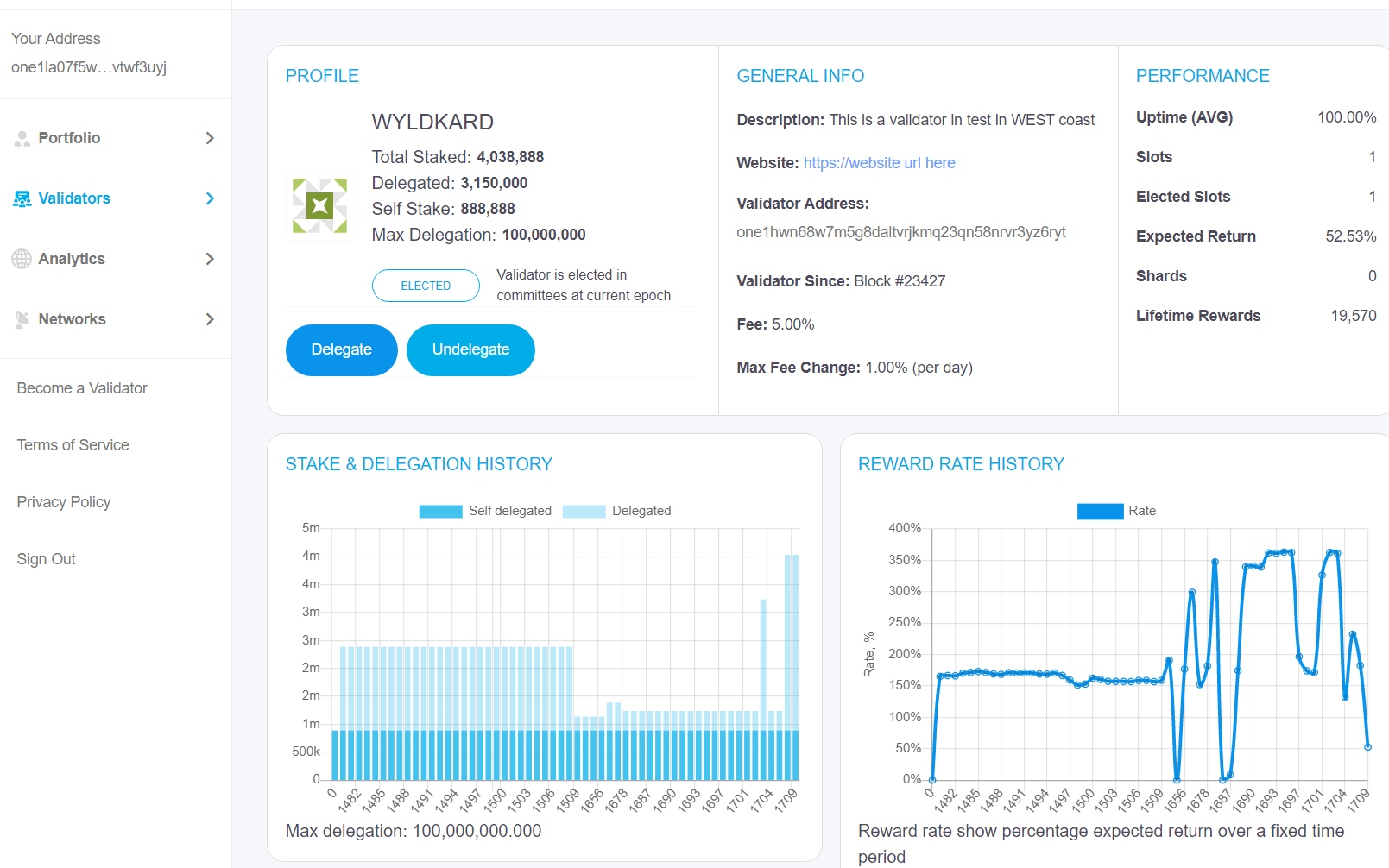


**Before attack**



**Attack description:** Through multiple delegations and undelegations to the same validator it is possible to manipulate the expected return of a validator so that stakers won't be interested to delegate to that respective validator.

**Result:** By using only a small amount of 3-4M ONE, it was possible to reduce the expected return from the highest of 361,2% to only 52,53%, and this artificially.



**Steps:**

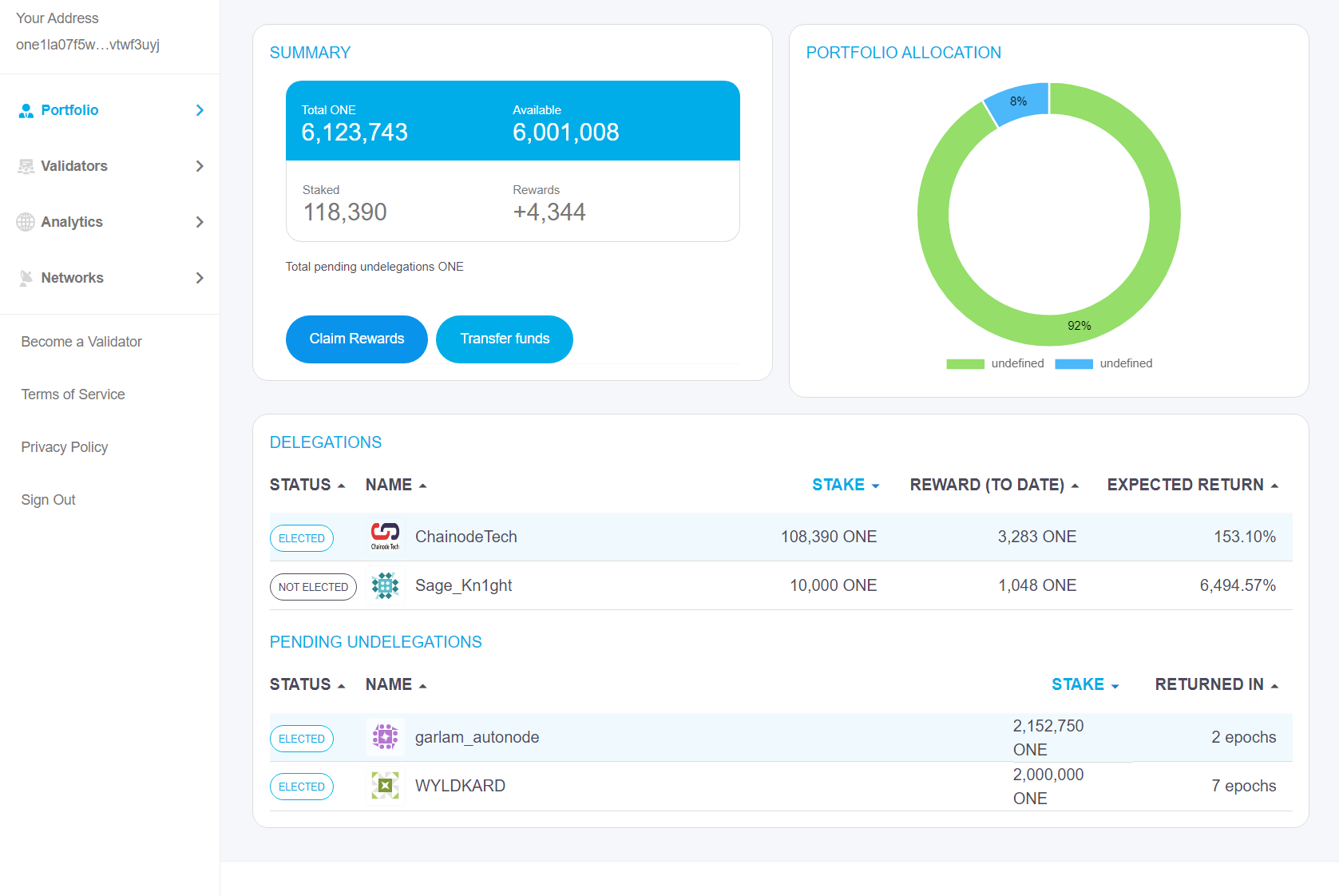
* Delegate an X amount to a validator;
* Undelegate the X amount;
* Delegate again a Z<X amount to the same validator;
* Repeat these steps several times.

The exploit used in this scenario is that it was actually possible to delegate - undelegate - redelegate to the same validator multiple times without having to wait for 7 epochs. This means that one attacker could use a relative small amount of ONE tokes to manipulate the real expected return of a validator, depending on its rank.

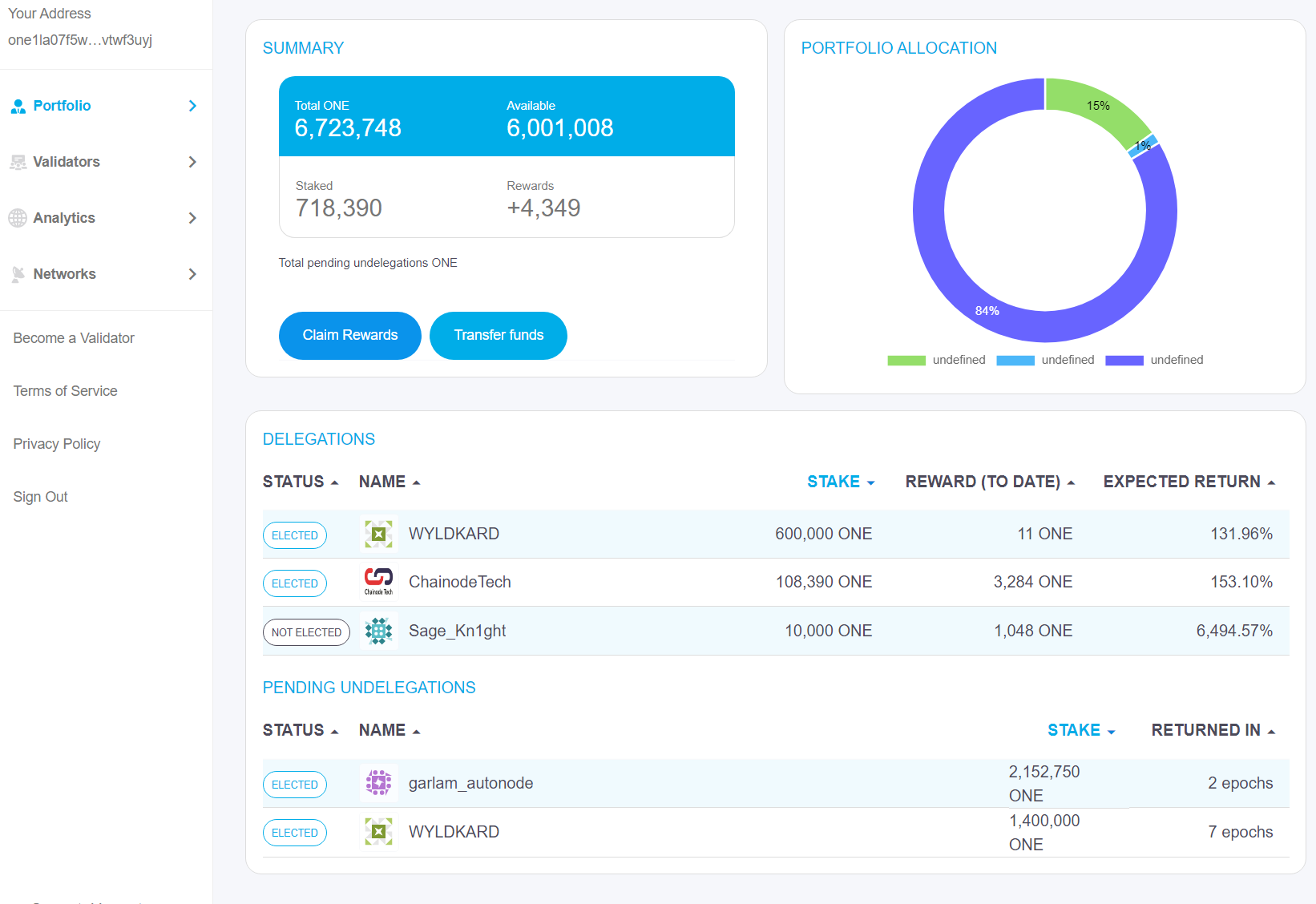
**Example:**

In the case below 2M ONE were delegated to WYLDKARD, then 2M ONE were undelegated and immediately after 600k ONE were delegated again to the same validator. As a result only 1,4M ONE remained under undelegations.

**2M ONE undelegation**



**Result after 600K redelegation to the same validator**

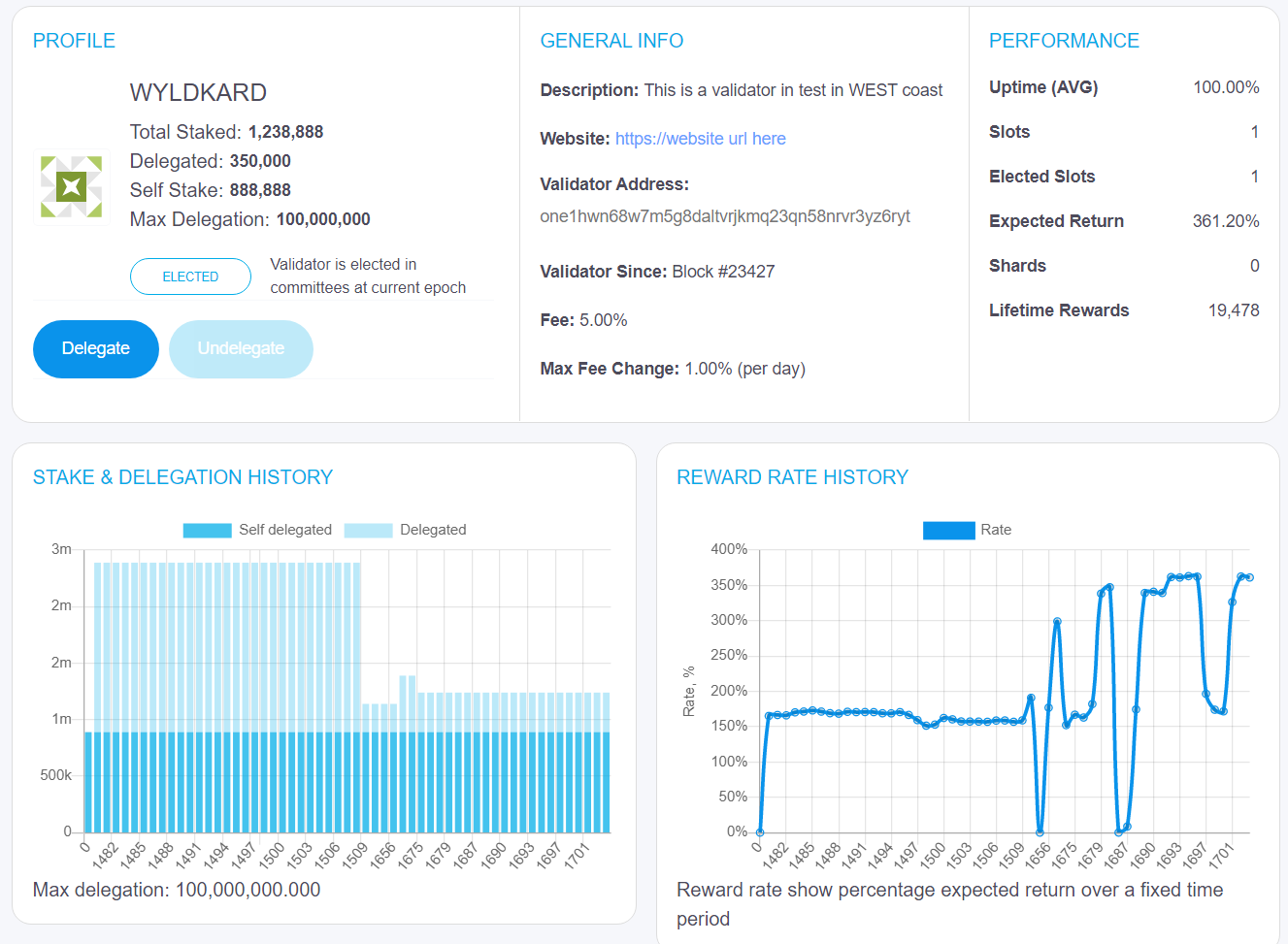


It also contradicts the philosophy of the tokens being undelegated after 7 epochs. By definition any action with those tokens should be possible only after 7 epochs have passed. I actually proposed a redelegation feature here <https://github.com/harmony-one/harmony/issues/2510> and generated numerous discussions with the conclusion not to implement it yet. But this is actually what I tried to exploit here, more like a particular case of the redelegation. I redelegated those tokens but to the same validator. So the current state of redelegation reflects more the possibility of an attack and has no other advantage, because if an user wants to undelegate from a validator then it won't want to redelegate immediately to the same validator.

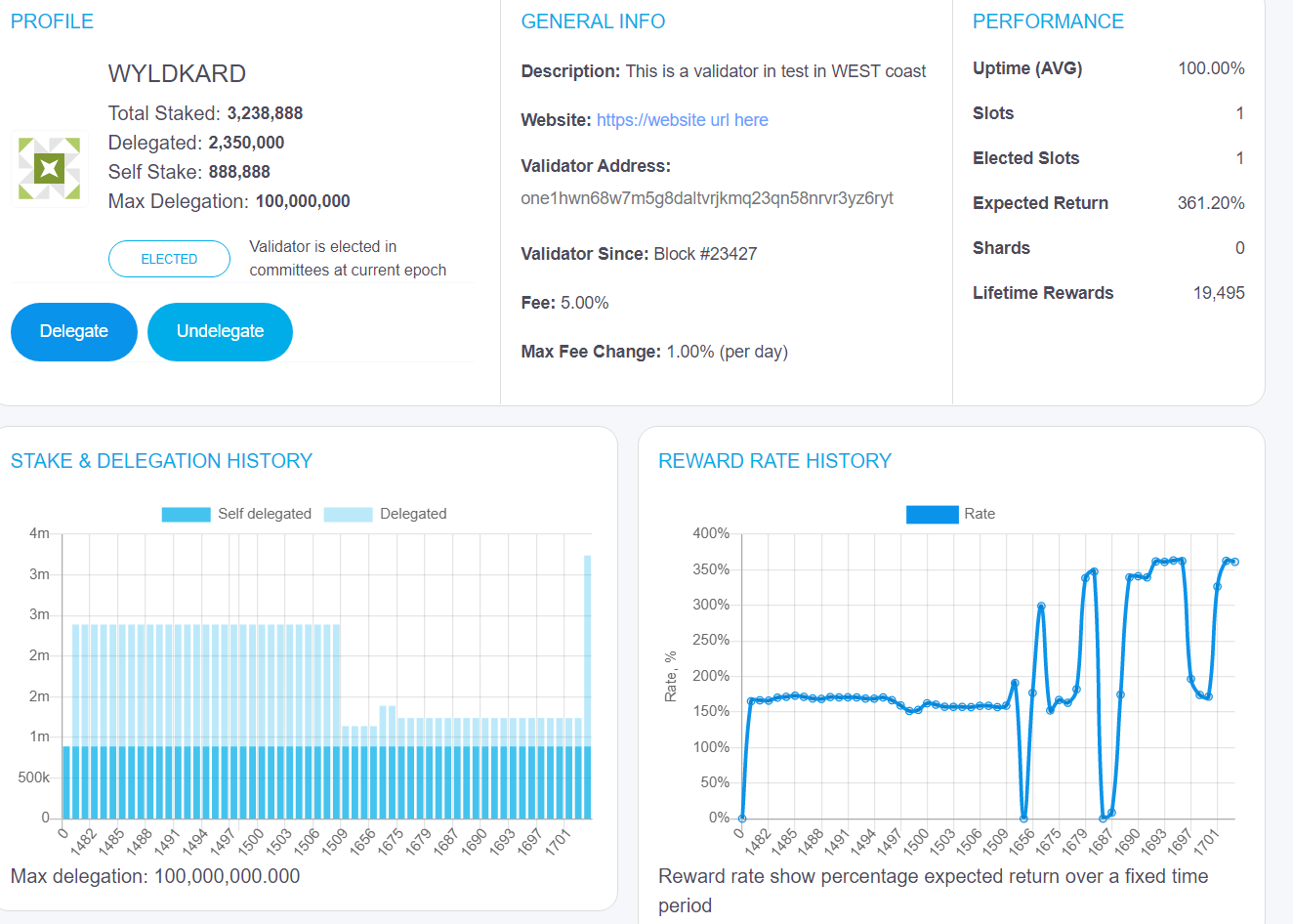
**Possible mitigation:** An easy mitigation which would also be conform to the EPoS philosophy is that all tokens are available for an action only after 7 epochs have passed. Therefore if a staker wants to redelegate to the same validator, it should be able to do this only after 7 epochs have passed. With this mitigation, if an attacker will want to manipulate the expected return of a validator continuously it will always need new tokens as the existing ones are locked for 7 epochs.

**Scenario 3:** Test multiple delegations/undelegations towards a validator to check if it remains elected during this process.

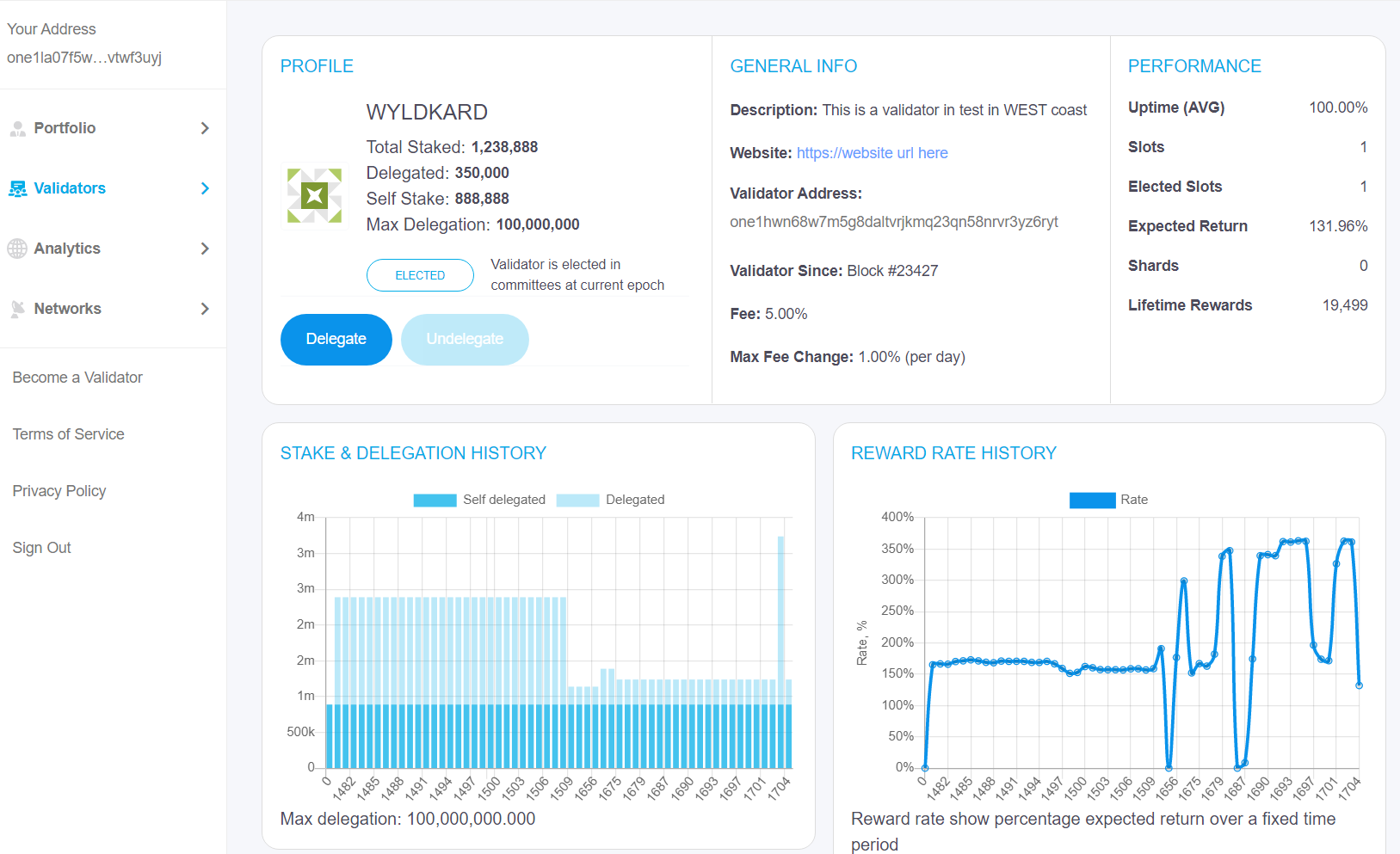
**Before delegation**



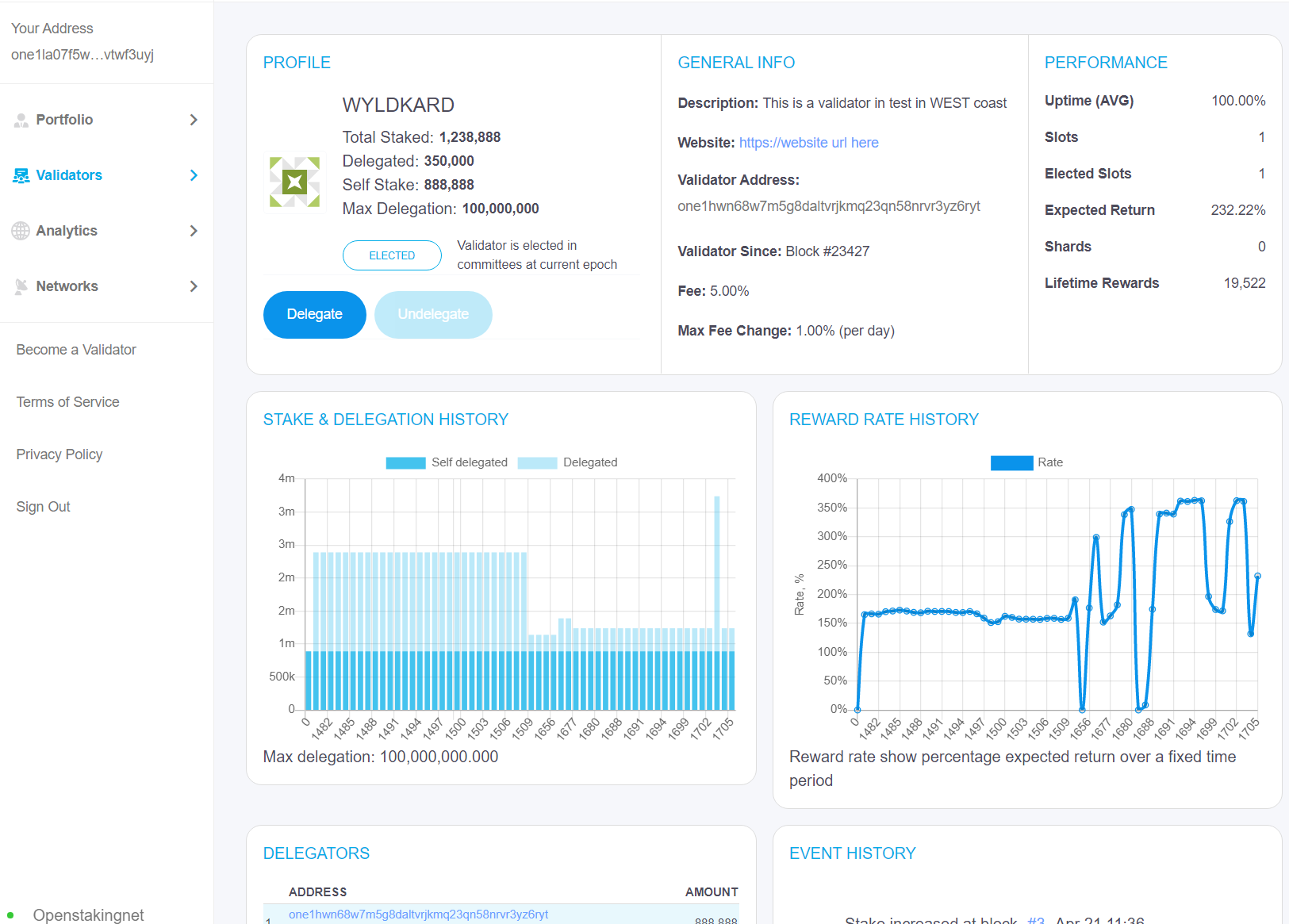
**2M ONE delegation -> OK**



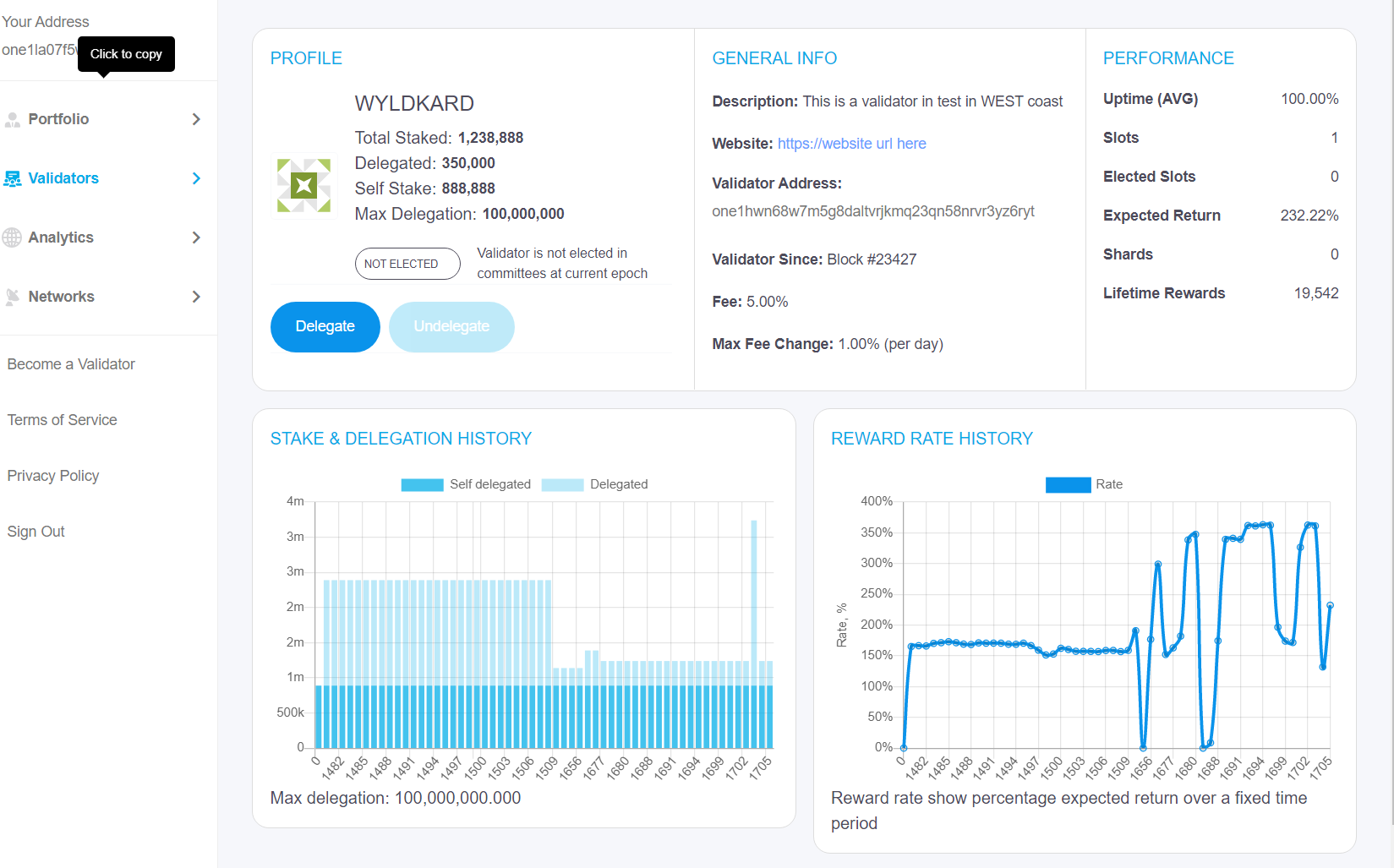
**2M undelegation -> OK**



**3M ONE delegation -> OK (in the screenshot the delegation doesn’t appear yet)**



**3M undelegation** -> validator changed to “Not Elected” state, **NOK**; The validator should still remain elected as it has enough delegation to be elected. It was previously elected with 1,23M and also worked for 2M ONE delegation and undelegation. Looking retrospectively to this problem it could also be that undelegation happened very fast after the delegation tx as it was also not available on the screen when I was doing the undelegation (s. screen above). But even then with 1,23M the validator was actually eligible for election and should be elected.



I tried this on garlam autonode as well and was able to put it on “Not Elected” status after delegation/undelegation and repetition of this.