

Security Audit

of the Energy Web Blockchain

June 12, 2019









Produced for



by



Table Of Contents

Foreword	1
Executive Summary	1
Audit Overview	2
1. Scope of the Audit	2
2. Depth of the Audit	2
3. Terminology	3
4. Limitations	3
System Overview	4
1. Holding	4
2. Validator	4
3. Reward	4
4. Registry	4
5. NodeControl	4
Best Practices in ENERGY WEB FOUNDATION's project	5
1. Hard Requirements	5
2. Best Practices	5
3. Test Suite	6
Security Issues	7
1. Reverse entries in SimpleRegistry could be deleted  	7
Trust Issues	8
Design Issues	9
1. Unnesesary pragma experimental ABIEncoderV2  	9
2. Transfer of reserved name to 0x0 allowed  	9
3. Unnesesary iteration through migrationValidators  	9

4.	Fee in SimpleRegistry to reserve a name	L	✓ Fixed	9
5.	Documentation mismatch with implementation	L	✓ Fixed	10
6.	Use keccak256 instead of sha256	L	✓ Fixed	10
7.	Validator can always set updateConfirmed to current block number	L	✓ Fixed	11
8.	Owner may not be a single entity across different contracts	L	✓ Acknowledged	11
9.	updateValidator does not validate arguments	L	✓ Fixed	11
	Recommendations / Suggestions			12
	Addendum and general considerations			13
1.	Pragma experimental ABIEncoderV2			13
2.	Reward is not exactly equal to 10 millions			13
3.	Community Fund not rewarded when address == 0x0			13
4.	Silent failure of contract creation in the genesis block			13
5.	Potential attacks against validator removal			14
6.	Outdated compiler version			14
7.	Forcing ETH into a smart contract			14
8.	Rounding Errors			14
	Disclaimer			15

Foreword

We would first and foremost like to thank ENERGY WEB FOUNDATION for giving us the opportunity to audit their project. This document outlines our methodology, limitations and results.

– ChainSecurity

Executive Summary

ENERGY WEB FOUNDATION engaged CHAINSECURITY to perform a security audit of ENERGY WEB BLOCKCHAIN, an Ethereum-based system.

ENERGY WEB FOUNDATION aims to provide an open-source, blockchain-based digital infrastructure for the energy sector based on the proven Ethereum implementation.

The EWT token represents the intrinsic currency in the system.

CHAINSECURITY audited the ENERGY WEB BLOCKCHAIN which is going to be deployed publicly. Audits of CHAINSECURITY use state-of-the-art tools for detection of generic vulnerabilities and checks of custom functional requirements. Additionally, a thorough manual code review by leading experts helps to ensure the highest security standards.

During the audit, CHAINSECURITY was able to help ENERGY WEB FOUNDATION in addressing a security and several design issues of low severity.

The employed coding practices are up to a high standard and the provided documentation and specification of the system are very detailed, which helps to ensure the intended functionality.

All reported issues have been addressed by ENERGY WEB FOUNDATION. In particular, all security and design issues have been eliminated with appropriate code fixes.

Audit Overview

Scope of the Audit

The scope of the audit is limited to the following source code files. They have been reviewed based on SOLC compiler, version 0.5.8 and EVM version PETERSBURG. All of these source code files were received on May 7, 2019 as part of the git commits 67be955c68d219b2b661311b63f17f4878016b08¹, 5e04f76af7c47bc37103eb94d1d9e7e04a28f3ce3² and 23db77e475ea01f9129a9bf1f3c2a1749370fa5a³. The latest update has been received on June 8, 2019 with the commit 19b8df749c445ad3787a07bf8fd5dfa0f17ef3e7⁴ and ac22c10d7b824255031b16b5702fc6ddecba4f5c9⁵.

In Scope	File	SHA-256 checksum
<input checked="" type="checkbox"/>	ewc-system-contracts/registry/Registry.sol	f5a229df8001776543bf62df9a9c890ff6548c67b731f94a726d5b18221ba4c5
<input checked="" type="checkbox"/>	ewc-system-contracts/reward/BlockReward.sol	d99afa2ce839940a0bfd8f0b38a0791adbd78f9acbf70d253fe6aa19bcee203d8
<input checked="" type="checkbox"/>	ewc-system-contracts/nodecontrol/NodeControlLookUp.sol	ee1f68c041e1c5b0eb58f5027daccb7b8fa2af1db3bebc68fc5b2fc1b6668e
<input checked="" type="checkbox"/>	ewc-system-contracts/interfaces/IValidatorSet.sol	037d1a3e2ac6f9b59033fcd5a5587bb1ba2033b423c1da7866a28f215ce96c5a
<input checked="" type="checkbox"/>	ewc-system-contracts/validatorset/ValidatorSetRelay.sol	7de198193227f2353054fc4298c4ec730f8f8b2bc33beb7f13b3ebca63e68c0e
<input checked="" type="checkbox"/>	.ewc-system-contracts/misc/Ownable.sol	961db40c3000280ee0056e09963bcc941575ea30b50bf95242086159df0f1060
<input checked="" type="checkbox"/>	ewc-system-contracts/nodecontrol/NodeControlDb.sol	741520da0617d1fcd205981aad244bb87fc51342192eda951f2a3ce90a4fa23
<input checked="" type="checkbox"/>	ewc-system-contracts/vesting/Holding.sol	9d091d314666dffdd6366865d5e0c34089abdf81ec83599dc2055fbd812e00a
<input checked="" type="checkbox"/>	ewc-system-contracts/interfaces/IValidatorSetRelayed.sol	a17900e1e2d47e5272ab6e003a7b0e47c08c8db38cceb9d20091b8f8396a3
<input checked="" type="checkbox"/>	ewc-system-contracts/interfaces/IValidatorSetRelay.sol	7de198193227f2353054fc4298c4ec730f8f8b2bc33beb7f13b3ebca63e68c0e
<input checked="" type="checkbox"/>	ewc-system-contracts/validatorset/ValidatorSetRelayed.sol	aeb8817e7bd6a9853f565de13bcf10aa44010893ddae446afa86054790c73f7e
<input checked="" type="checkbox"/>	ewc-system-contracts/reward/SCurveProvider.sol	5a6e53cf2b60f4a4297fe3a466b5ad5a4d4ed37bb40e88891cc49c8e34422
<input checked="" type="checkbox"/>	ewc-system-contracts/registry/SimpleRegistry.sol	ad8e6b938be90eb87256f4d0137206b4773d2e55450e9e393eb0df35adca1476
<input checked="" type="checkbox"/>	ewc-system-contracts/interfaces/INodeControl.sol	687f5d47d0ee9fff05ac6fb0c77ec915cd9d8accf159a67696af3a6682c9782
<input checked="" type="checkbox"/>	ewc-system-contracts/interfaces/IBlockReward.sol	981c6ce7eb3a3bef7374b98ab851958324e5e605b9b35b8e9ffedf59c0d18057
<input checked="" type="checkbox"/>	ewc-system-contracts/libs/SafeMath.sol	571d21a9d92def1a55fade7167f36a5fb110ad9e649cb9f38046a0fb9236de1e
<input checked="" type="checkbox"/>	ewc-system-contracts/nodecontrol/NodeControlSimple.sol	0f697c0f927c575fa17e0de0ec1f9be944f0f66593196a312b1118a629058b66
<input checked="" type="checkbox"/>	ewf-chainspec/Volta.json	6c83b3f61d171d2742f8f100c2aa95318ef51de336dcadfedc9785e628307998
<input checked="" type="checkbox"/>	ewf-genesis-generator/index.js	785d4841a199ad452e84065540125c48831d5111de9dd3138faf56b0e34accef
<input checked="" type="checkbox"/>	ewf-genesis-generator/chainspec_skeletons/volta.json	11487f2ebb44caeb4e265774c85010f3aab2c40195a6c7e7c24e8be2a70696f
<input checked="" type="checkbox"/>	ewf-genesis-generator/build.sh	843f681c193b9cb01835c1cf712a45973d76918bf56bd769f1b40d386cf1d9f1
<input checked="" type="checkbox"/>	ewf-genesis-generator/chainspec_skeletons/hardcoded_values_volta.json	a0fdfb32f86560b1a7728ffdeca2b565d8b73d610ca4a871af6799e229499de

For these files the following categories of issues were considered:

In Scope	Issue Category	Description
<input checked="" type="checkbox"/>	Security Issues	Code vulnerabilities exploitable by malicious transactions
<input checked="" type="checkbox"/>	Trust Issues	Potential issues due to actors with excessive rights to critical functions
<input checked="" type="checkbox"/>	Design Issues	Implementation and design choices that do not conform to best practices

Depth of the Audit

The scope of the security audit conducted by CHAINSECURITY was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

¹<https://github.com/energywebfoundation/ewc-system-contracts/tree/67be955c68d219b2b661311b63f17f4878016b08>

²<https://github.com/energywebfoundation/ewf-genesis-generator/tree/5e04f76af7c47bc37103eb94d1d9e7e04a28f3ce3>

³<https://github.com/energywebfoundation/ewf-chainspec/tree/23db77e475ea01f9129a9bf1f3c2a1749370fa5a>

⁴<https://github.com/energywebfoundation/ewc-system-contracts/tree/19b8df749c445ad3787a07bf8fd5dfa0f17ef3e7>

⁵<https://github.com/energywebfoundation/ewf-genesis-generator/tree/ac22c10d7b824255031b16b5702fc6ddecba4f5c9>

Terminology





For the purpose of this audit, CHAINSECURITY has adopted the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology⁶).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.










Impact specifies the technical and business-related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.





We categorise the findings into four distinct categories, depending on their severities:



-  Low: can be considered less important
-  Medium: should be fixed
-  High: we strongly suggest fixing it before release
-  Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the table below, following a standard approach in risk assessment.

LIKELIHOOD	IMPACT		
	High	Medium	Low
High			
Medium			
Low			

During the audit, concerns might arise or tools might flag certain security issues. After carefully inspecting the potential security impact, we assign the following labels:

-  **No Issue**: no security impact
-  **Fixed**: the issue is addressed technically, for example by changing the source code
-  **Addressed**: the issue is mitigated non-technically, for example by improving the user documentation and specification
-  **Acknowledged**: the issue is acknowledged and it is decided to be ignored, for example due to conflicting requirements or other trade-offs in the system

Findings that are labelled as either  **Fixed** or  **Addressed** are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview of what kind of issues were found during the audit.

Limitations

Security auditing cannot uncover all existing vulnerabilities; even an audit in which no vulnerabilities are found is not a guarantee of a secure smart contract. However, auditing enables discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. This is why we carry out a source code review aimed at determining all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

⁶https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

System Overview

ENERGY WEB FOUNDATION deploys an Ethereum-based blockchain called Volta where applications of the energy sector are intended to run on. Smart contracts compiled to bytecode compatible with EVM version Petersburg are supported. Consensus is based on Proof-of-Authority, where validators seal new blocks in a round-robin manner. The list of the currently active validators is governed by a smart contract which is fully controlled by the ENERGY WEB FOUNDATION. Trusted parties of the energy sector, partners of the ENERGY WEB FOUNDATION fulfilling certain defined criteria are eligible to be added as validators. Misbehaving validators can be reported and removed by the ENERGY WEB FOUNDATION.

The Parity client with its support for AURA-based POA is to be used to run the network. The Genesis Block defines and deploys the essential smart contracts at certain predefined addresses, detailed below. For upgradeability, some of these contracts are implemented with a proxy contract relaying calls to an exchangeable contract implementing the logic, called relay-relayed pattern in the ENERGY WEB BLOCKCHAIN.

The ENERGY WEB BLOCKCHAIN smart contract suite consists of multiple independent parts:

Holding

The holding contract serves as vesting contract to lock up funds for certain hardcoded addresses, amounts and durations. Once the lock up time has been exceeded, the funds can be released.

Validator

These contracts govern the Proof-of-Authority, maintain the list of current validators and provide interfaces for interactions with the Parity client. The system acts through `ValidatorSetRelay` which forwards the calls to `ValidatorSetRelayed`. `ValidatorSetRelayed` manages the list of active `Validators` and provides functionality to query and change its entries. All state changing actions can only be invoked either through the owner or the relay contract. Provides functionality for `Validators` to report validator as benign or malicious.

Reward

Handles the block reward functionality. The Blockchain client queries `reward()` at every new block which returns a list of addresses and the amount they are to be rewarded with. Logging functionality keeping track of the total reward paid out is implemented. The block reward is based on an `SCurve` scheme, where the reward to be paid out reduces over a period of 10 years and becomes 0 afterwards. Any user may choose to set a payout address, however only validators get payouts after successfully sealing a block.

Registry

This is Parity's name registry smart contract enhanced with ownership functionality. ENERGY WEB FOUNDATION states it might be needed for system related contracts later, this contract is only intended for internal purposes. For the public an official ENS system will be setup.

NodeControl

These smart contracts interact with ENERGY WEB BLOCKCHAIN's node control client (via Events) that must be installed on validator nodes next to the Parity clients. These events can signal updates to update the used docker image or download new chain specifications. Validators can call a function of the smart contract to confirm they've updated.

Furthermore following externally owned accounts have a pre-allocated balance:

0x0cB1437200aea736788f1Fc56F327c0456c3598D:	"balance": "25000000000000000000"
0x74dd76E24B2CFB43C1b1a4498295d553D0843746:	"balance": "25000000000000000000"
0xeeB4CEe443F9e0D17BdBd6Daa241681EE5E51c2:	"balance": "25000000000000000000"
0xA005caEa55375ae20e3aAEF746113535503ABC19:	"balance": "25000000000000000000"

Best Practices in ENERGY WEB FOUNDATION's project

CHAINSECURITY is determined to deliver the best results to ensure the security of a project. To enable us to do so, we are listing Hard Requirements which must be fulfilled to allow us to start the audit. Furthermore we are providing a list of proven best practices. Following them will make audits more meaningful by allowing efforts to be focused on subtle and project-specific issues rather than the fulfilment of general guidelines.

Hard Requirements

These requirements ensure that the ENERGY WEB FOUNDATION's project can be audited by CHAINSECURITY.

- ☒ **All files and software for the audit have been provided to CHAINSECURITY**
The project needs to be complete. Code must be frozen and the relevant commit or files must have been sent to CHAINSECURITY. All third party code (like libraries) and third-party software (like the solidity compiler) must be exactly specified or made available. Third party code can be located in a folder separated from client code (and the separation needs to be clear) or included as dependencies. If dependencies are used, the version(s) need to be fixed.
- ☒ The code must compile and the required compiler version must be specified. When using outdated versions with known issues, clear reasons for using these versions are being provided.
- ☒ There are migration/deployment scripts executable by CHAINSECURITY and their use is documented.
- ☒ The code is provided as a Git repository to allow reviewing of future code changes.

Best Practices

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable.

- ☒ There are no compiler warnings, or warnings are documented.
- ☒ Code duplication is minimal, or justified and documented.
- ☒ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ☒ The project only contains audit-related files, or, if this is not possible, a meaningful distinction is made between modules that have to be audited and modules that CHAINSECURITY should assume are correct and out-of-scope.
- ☒ There is no dead code.
- ☒ The code is well-documented.
- ☒ The high-level specification is thorough and enables a quick understanding of the project without any need to look at the code.
- ☒ Both the code documentation and the high-level specification are up-to-date with respect to the code version CHAINSECURITY audits.
- ☒ Functions are grouped together according to either the Solidity guidelines⁷, or to their functionality.

⁷<https://solidity.readthedocs.io/en/v0.4.24/style-guide.html#order-of-functions>

Test Suite

In this section, CHAINSECURITY comments on the test suite of ENERGY WEB BLOCKCHAIN. While the test suite is not a component of the audit, a good test suite is likely to result in better code.

All smart contracts were provided with extensive test cases and appear to cover all intended use cases. The provided test cases also cover failure cases. The tests contain proper assertions to check for the correct error messages making sure that errors were handled as expected.

However CHAINSECURITY wants to note that the tests are not related to the actual migration/use of the smart contracts. The audited smart contracts are going to be deployed during the genesis block of the ENERGY WEB FOUNDATION blockchain and are a vital part of this system. The test suite provided however deployed and tested the smart contract as normal smart contracts. No test provided deployed the smart contract as part of the genesis block of a proof-of-authority chain and thoroughly checked their proper functionality.

Security Issues

This section relates to our investigation into security issues. It is meant to highlight times when we found specific issues, but also mentions what vulnerability classes do not appear, if relevant.

Reverse entries in SimpleRegistry could be deleted

The Parity name registry contract stores a bi-directional mapping from addresses to strings. This provides a simple mechanism to look up addresses using regular names and to reverse-lookup addresses. Such a reverse-lookup entry can be deleted by an attacker, if he is the owner of any other name, meaning he satisfies the requirement `require(entries[_name].owner == msg.sender)` for at least one `_name`.

The ENERGY WEB FOUNDATION uses a slightly modified version of Parity's name registry contract, called SimpleRegistry. The implementation was extended by adding OpenZeppelin's Ownable functionality. Furthermore, the `onlyOwner` modifier was added to the `reserve` function, meaning only the owner may reserve a name in ENERGY WEB FOUNDATION's implementation. For this attack, the attacker ownership of a name, e.g. by a `reserve()` and `transfer()` done by the owner.

We assume a properly setup entry of an arbitrary victim entry, where:

- `victimEntry` has been reserved using `reserve(victimEntry)`
- `proposeReverse(victimName, victimReverse)` has been executed and been properly confirmed by `victimReverse` by calling `confirmReverse(victimName)`.

For such a valid entry, an attacker can delete the reverse-lookup (`reverses[victimReverse]`, which currently stores `victimName`) by completing the following steps:

1. Obtain ownership of any entry - in the original implementation this is possible by calling `reserve(attackerEntry)`, in ENERGY WEB FOUNDATION's implementation the attacker needs to convince the owner to transfer the ownership of any name.
2. `proposeReverse(attackerName, victimReverse)`
3. `drop(attackerEntry)`

The call to `drop()` executes `delete reverses[entries[_name].reverse];` where `entries[_name].reverse` has previously in step 2) been set to the target of the victim. Note this is possible without the need to execute `confirmReverse()` where the target address for the reverse needs to call to confirm, which would inhibit this attack.

Overall, this bug has a low likelihood in ENERGY WEB FOUNDATION's system, given the modifications and the intended use, but has a higher likelihood in the actual name-registry implementation of Parity where anyone can reserve a name by paying the fee.

Likelihood: Low

Impact: Medium

Fixed: ENERGY WEB FOUNDATION solved the issue by adding an additional check inside the `drop` function to ensure that `keccak256(bytes(reverses[entries[_name].reverse]))` equals the passed `_name`. Additionally dropped entries are not lost forever but can be registered again by their owner.

Trust Issues

This section mentions functionality that is not fixed inside the smart contract and hence requires additional trust in ENERGY WEB FOUNDATION, including in ENERGY WEB FOUNDATION's ability to deal with such powers appropriately.

CHAINSECURITY has no concerns to raise in this category of the report.

Design Issues

This section lists general recommendations about the design and style of ENERGY WEB FOUNDATION's project. These recommendations highlight possible ways for ENERGY WEB FOUNDATION to improve the code further.

Unnessesary pragma experimental ABIEncoderV2

NodeControlLookUp.sol declares `pragma experimental ABIEncoderV2` however does not use any of the new functionality of the experimental ABIEncoderV2. CHAINSECURITY recommends to use the legacy ABIEncoder to mitigate the risk of encountering unexpected behaviour of experimental functionality if these are not explicitly needed.

Fixed: ENERGY WEB FOUNDATION removed `pragma experimental ABIEncoderV2` from NodeControlLookUp.sol.

Transfer of reserved name to 0x0 allowed

The SimpleRegistry contract features the modifier `whenUnreserved` which ensures that the Entry has no owner yet. This is done by comparing the current owner to `address(0)`. Only unreserved addresses may be reserved by calling the function `reserve()`.

Function transfer allows the current owner of an entry to transfer it to a new address. This function does not prohibit transfer of ownership to `address(0)`, thus a reserved address's owner can be set to `address(0)`. Consequently a valid, reserved entry with non-zero fields passes the `whenUnreserved` modifier and may be reserved again using `reserve()`.

Fixed: ENERGY WEB FOUNDATION added a restriction, it's not possible to transfer reserved names to 0x0 anymore.

Unnessesary iteration through migrationValidators

`finalizeChange()` of `ValidatorSetRelayed` is inefficient. The current implementation of `addValidator()` adds only ever one new validator at the last position, while `removeValidator()` removes the entry to be removed and assigns it to `toBeRemoved`.

Currently the implementation is inefficient and always iterates through `migrationValidators`. This is unnecessary:

1. only the last entry is new and needs to be handled
2. in case an address has to be removed, the whole iteration is unnecessary.

Fixed: ENERGY WEB FOUNDATION removed the iteration.

Fee in SimpleRegistry to reserve a name

A fee needs to be paid in order to register a name in the SimpleRegistry contract. The purpose of this fee is unclear, as only the Owner is allowed to reserve a name and the owner has full control over the funds held by this contract and can withdraw these at any time by calling `drain()`.

Additionally `reserve()` only ensures that `msg.value >= fee`, if by mistake a higher amount is sent along the transaction, this is simply accepted and kept by the contract.

Fixed: The fee to register a name, and all related functionality has been removed.

Documentation mismatch with implementation

Some minor differences between the specifications and the actual implementation were discovered:

Concerning the Validator set contracts:

- SYSTEM **address** can be changed by the owner

The implementation features no functionality for the owner to do so.

- Shall not enforce any lower on (<- contains typo, should be or) upper limit on the number of active validators

A lower limit of 1 is enforced upon deployment and when removing a validator. Furthermore this requirement conflicts with the requirement Shall enforce to **always** have at least 1 validator in the same section.

Concerning the Node control contracts:

- The **function** RetrieveUpdate (<- contains typo)... Must **return** the stateStruct containing the current validator state.

In the implementation, the function is called retrieveExpectedState.

Fixed: ENERGY WEB FOUNDATION updated to documentation, the specification document now reflects the actual implementation.

Use keccak256 instead of sha256

updateValidator() of the NodeControlSimple contract ensures that the new values actually contain a change. Due to limitations in Solidity, this is done by comparing the hashes of the values:

```
require(
    !(sha256(bytes(nodeControlDb.getDockeSha(_targetValidator))) ==
      sha256(bytes(_dockeSha)) && sha256(bytes(nodeControlDb.
        getDockeName(_targetValidator))) ==
      sha256(bytes(_dockeName)) && sha256(bytes(nodeControlDb.
        getChainSpecSha(_targetValidator))) ==
      sha256(bytes(_chainSpecSha)) && sha256(bytes(nodeControlDb.
        getChainSpecUrl(_targetValidator))) ==
      sha256(bytes(_chainSpecUrl)) && nodeControlDb.getIsSigning(
        _targetValidator) == _isSigning),
    "Error: No changes in the passed State"
);
```

Using the builtin keccak256 operation would be significantly more efficient compared to the use of **sha256** which is implemented as precompiled contract.

Contract: Testing gas costs for sha256 vs. keccak256
– Compiled with optimization enabled

[+] Testing short strings

(index)	sha256	keccak256	diff
codesize	605	428	177
deploymentGas	245511	198621	46890
usageGas	25302	23453	1849

[+] Testing one long string

(index)	sha256	keccak256	diff
codesize	605	428	177
deploymentGas	401265	354375	46890
usageGas	26958	25073	1885

Fixed: ENERGY WEB FOUNDATION replaced sha256 with keccak256.

Validator can always set updateConfirmed to current block number

A validator may call `confirmUpdate()` which sets `updateConfirmed` to the current blocknumber, even if there is currently no update to be confirmed.

Fixed: `updateConfirmed()` can now only be called successfully if there is actually an update to confirm.

Owner may not be a single entity across different contracts

Regarding ownership, the specification document states for both sets of contracts, `ValidatorSet` and `NodeControl` the following:

Must be owned by a single entity, referred to as owner

Note that multiple contracts are deployed, `ValidatorSetRelay` & `ValidatorSetRelayed` and `NodeControlDB`, `NodeControlLookUp` & `NodeControlSimple` respectively. Each deployed contract has its own implementation of `Ownable` thus their owners may differ. The OpenZeppelin implementation of `Ownable` is used to implement the ownership functionality. Note that there is a `renounceOwnership()` function where the current owner can renounce the ownership, effectively breaking the requirement of the specification document stated above.

Acknowledged: ENERGY WEB FOUNDATION will make sure through its management processes that each contract has the same entity as owner. It might be possible that different Multisigs would own different contracts for increased security, but all the owner Multisigs would be controlled by EWF, the owner entity. The `renounceOwnership` function has been removed from the `Ownable` contract.

updateValidator does not validate arguments

Function `updateValidator` present in the `NodeControlSimple` contract takes some parameters and updates the data for the target validator address.

However, there is no check present in the code to ensure that there are non-zero length data passed in these parameters, consequently some fields might be updated with zero-length data.

This gets problematic when a zero-length data field is taken as indicator that no valid entry exists. This is done frequently as in Ethereum a non-initialized storage entry defaults to 0. Function `confirmUpdate` does this to filter out transactions sent by non-validators:

```
function confirmUpdate()
    external
{
    require(nodeControlDb.getDockersha(msg.sender).length != 0, "Error:
        You are not a validator!");
    nodeControlDb.setUpdateConfirmed(msg.sender);
}
```

Due to the missing input validation a validator may have an empty zero-length entry set for the `Dockersha` entry, thus would fail to pass this `require` statement.

Fixed: ENERGY WEB FOUNDATION added input validation to check that the passed input parameters are of non-zero length. Certain fields however, like `dockersha` and `chainspecsha` have a specific length, input validation may be enhanced to ensure valid parameters are passed.

Recommendations / Suggestions

- ✓ The `NodeControlLookUp` contract stores the address of the current `nodeControlContract`. The function `changeAddress` allows the owner to change this address. CHAINSECURITY recommends to emit an event upon such a significant state change.

Potential attacks against validator removal

ENERGY WEB FOUNDATION controls the list of `currentValidators` in the `ValidatorSetRelayed` contract. The functions `addValidator` and `removeValidator` allow ENERGY WEB FOUNDATION to change these entries. To do so, ENERGY WEB FOUNDATION needs to craft a transaction calling this function and get this transaction included in a block. Note that a validator being removed from the list may ignore such a transaction. Colluding validators may do so successfully over multiple blocks. Even if validators do not cooperate, a validator to be removed may try to fill up the next blocks with gas-consuming transactions to prevent this transaction to be included. This might be successful with high probability if he chooses a higher gas price. Given that the block reward is considerable and depending on the gas price, this DoS might be economically beneficial for such a validator.

Outdated compiler version

CHAINSECURITY could not find obvious issues with the compiler version ENERGY WEB FOUNDATION is using. While the latest compiler release is version 0.5.9, ENERGY WEB FOUNDATION uses version SOLC compiler, version 0.5.8. If ENERGY WEB FOUNDATION is aware of the compiler's behavior and bugs, there might be good reasons for using an older compiler version. While the latest version does contain bug fixes, it might introduce new bugs.

CHAINSECURITY does, however, recommend to use the same compiler version homogeneously throughout the project and to use the compiler version for deployment that was used during testing. Furthermore, for any used version it is helpful to monitor the list of known bugs⁹.

Forcing ETH into a smart contract

Regular EWT transfers to smart contracts can be blocked by those smart contracts. On the high-level this happens if the according solidity function is not marked as **payable**. However, on the EVM levels there exist different techniques to transfer EWT in unblockable ways, e.g. through **selfdestruct** in another contracts. Therefore, many contracts might theoretically observe "locked EWT", meaning that EWT cannot leave the smart contract any more. In most of these cases, it provides no advantage to the attacker and is therefore not classified as an issue.

Rounding Errors

(Unsigned) integer divisions generally suffer from rounding errors. The same holds true for divisions inside the EVM. Therefore, the results of arithmetic operations can be imprecise. The effects of these errors can be reduced by ordering arithmetic operations in a numerically stable manner. However, even then minor errors (e.g. in the order of one token wei) can occur.

⁹<https://solidity.readthedocs.io/en/develop/bugs.html>

Disclaimer

UPON REQUEST BY ENERGY WEB FOUNDATION, CHAINSECURITY LTD. AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND CHAINSECURITY LTD. DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH CHAINSECURITY LTD..