NEAR Protocol

NEARCore Security Review #2

Version: 1.0

January, 2020

Contents

Introduction

NEAR is a blockchain leveraging a proof-of-stake consensus algorithm and a sharding design to address some of the scalability and performance challenges faced by existing blockchain networks.

Sigma Prime performed an initial security review (in Q4 2019) targeting NEARCore, the official reference implementation of NEAR, developed in Rust.

Sigma Prime was commercially engaged to perform a new time-boxed security asssessment of NEARCore, after several changes have been introduced by the development team.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the software assessed. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of NEARCore contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation.

Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Finally, the Appendix section provides additional documentation, including the severity matrix used to classify vulnerabilities within the NEARCore Blockchain client.

Overview

NEARCore is the official implementation of NEAR protocol, a next-generation blockchain network focussed on sharding, developed in Rust.

NEAR allows developers to write, test and deploy scalable and decentralized applications using existing high-level programming languages that compile down to WebAssembly, such as Typescript and Rust, using `near-bindgen`, a Rust library for writing smart contracts.

NEAR also provides an online Integrated Development Environment (IDE), called NEAR Studio, enabling developers to leverage an embedded deployment process.

Security Assessment Summary

Assessment Scope

This review was initially conducted on the following commits:

- `nearcore` : commit d781c41
- `near-bindgen` : commit 61b0ef9

This security assessment targeted the following components of these repositories:

- `nearcore/chain` : Defines the NEAR blockchain logic (types, peer-to-peer stack, serialisation, JSON-RPC definition, syncing mechanism, etc.);
- `nearcore/core` : Contains the core components used by the NEAR nodes, such as cryptographic primitives (i.e. Ed25519 signing), persistent storage functions, merklization helper functions, etc;
- `near-bindgen/near-bindgen` : Library allowing the development of WASM smart contracts in Rust;
- `near-bindgen/near-bindgen-core` : Core part of the library for writing NEAR smart contracts;
- `near-bindgen/near-bindgen-macros` : Main macro of the library for writing NEAR smart contracts;
- `near-bindgen/near-bindgen-promise` : Implementation of cross-contract wrappers through macros.

Additionally, this review also targeted NEAR's Google Cloud Platform (GCP) infrastructure.

Assessment Methodology Overview

The testing team performed a time-boxed manual review of the relevant codebases (see repositories above) and cloud services, including:

- **RPC Service:** https://rpc.nearprotocol.com/;
- **NEAR Wallet:** https://wallet.nearprotocol.com/;
- **NEAR Explorer:** https://explorer.nearprotocol.com/.

Additionally, the testing team leveraged fuzz testing techniques (i.e. fuzzing), which is a process that allows the identification of bugs by providing randomised and unexpected data inputs to software with the purpose of causing crashes (or in Rust, *panics*) and other unexpected behaviours (e.g. memory leaks).

Sigma Prime produced fuzzing targets targeting NEARCore using libFuzzer, a library part of the LLVM project, enabling coverage-guided fuzz testing. This library focuses on:

- **In-process fuzzing**: the fuzzing engine executes the target many times with multiple data inputs in the same process. It must tolerate any kind of input (empty, huge, malformed, etc);

- **White-box fuzzing**: libFuzzer leverages compiler instrumentation and requires access to the source code;
- **Coverage-guided fuzzing**: for every input/test case, libFuzzer tracks code paths (sections of the code reached), and produces variants of each test case to generate additional input data to increase code coverage.

In addition to these fuzzing targets, the testing team developed several tests written for the purpose of this assessment, located inside the `tests/tests` folder:

- `attribute-ordering`
- `callback`
- `callback-and-args`
- `callback-args-bad-order`
- `callback-args-empty`
- `clone-promise`
- `collections-static`
- `double-promise`
- `empty-borsh`
- `fnarg-type`
- `fnarg-type-box`
- `init-twice`
- `iterator-empty`
- `iterator-overlap`
- `max-storage`
- `mod-overlap`
- `no-init`
- `non-serde-args`
- `redeploy`
- `state-growth`
- `state-overwrite`

During the course of this assessment the testing team identified a total of nineteen (19) issues during this assessment, of which:

- Two (2) are classified as critical risk,
- Three (3) are classified as high risk,
- One (1) is classified as medium risk,
- Eight (8) are classified as low risk,
- Five (5) are classified as informational.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the the scope of this assessment. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
NEAR-01	Initialisation Function Can Be Called Multiple Times	Critical	Open
NEAR-02	Block Censorship	Critical	Open
NEAR-03	Crate Mutable Variables Are Not Stored On-chain	High	Open
NEAR-04	Overwriting Contract State Storage	High	Open
NEAR-05	Insecure Storage of Sensitive Information	High	Open
NEAR-06	Block Production Speedup	Medium	Open
NEAR-07	Initialisation Function Incorrectly Set	Low	Open
NEAR-08	Collisions within `Collections`	Low	Open
NEAR-09	Overexpansion of Contract State	Low	Open
NEAR-10	Accuracy of Reward Calculation Variables	Low	Open
NEAR-11	API Does not Enforce Strict Transport Security (HSTS)	Low	Open
NEAR-12	API Does not Enforce Strict Transport Security (HSTS)	Low	Open
NEAR-13	Weak TLS Protocols and Ciphers Supported	Low	Open
NEAR-14	Insufficient Protection Against Clickjacking	Low	Open
NEAR-15	Receiver Argument Type	Informational	Open
NEAR-16	Finalisation Fork	Informational	Open
NEAR-17	General Design Comments	Informational	Open
NEAR-18	Denial-of-Service Protection not in Use (Cloud Armor)	Informational	Open
NEAR-19	Google Cloud Access Transparency Disabled	Informational	Open

NEAR-01	Initialisation Function Can Be Called Multiple Times		
Asset	`near-bindgen`		
Status	**Open**		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

An initialisation function can be set in the `near_bindgen` macro allowing the user to write code to produce the contract state. The initialisation function can optionally be assigned to a public function within the implementation.

An assigned function is public and may be called multiple times. Each successive call will overwrite the contract state according to the initialisation function.

For a proof-of-concept see test case `tests/init-twice`.

Recommendations

Developers are able to ensure an initialisation function can only be called once within the contract bytecode. However we recommend including an initialisation flag that only allows the init function to be called once.

NEAR-02	Block Censorship		
Asset	`nearcore`		
Status	**Open**		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

A block's weight is calculated by $weight(n) = weight(n-1) + approved_stake/total_stake * (t(n-1) - t(n-2))$. Where $t(i)$ refers to the timestamp of block i and $approved_stake$ is the sum of the stake of each validator who has attested.

Starting at $block(i)$ the next honest block producer will creates block $block(i+1)$. However, if a malicious user is the next block producer and wants to censor the previous block, they may create a fork from $block(i)$, call this $block(i+x)$. The malicious user includes all of the attestations in $block(i+1)$ and one more (for examplel, their own if they wish).

Only one variable in the weight equation $approved_stake$ has changed and it increased, as all previous attestations are included plus one additional one. Thus, $weight(i+x) > weight(i+1)$. Honest validators will now consider $block(i+x)$ to be the new head and continue to build off this block thereby censoring the previous block.

The attack can also be applied to perform large block skipping. Block validation ensures that the current block height has not increased by more than `expected_epoch_length`. As a result, an attacker that is scheduled to produce a block within the next `expected_epoch_length` number of blocks can produce a block to apply the attack above and becoming the head.

Mass block skipping can be applied regularly and impacts the minting of NEAR tokens. As minting occurs at the end of every epoch this attack reduces the amount of time before the end of the epoch is reached and as a result increases the amount of NEAR that is minted per year.

Recommendation

The attack vector described above is known by the development team and will be taken into consideration in the new fork choice design.

NEAR-03	Crate Mutable Variables Are Not Stored On-chain		
Asset	`near-bindgen`		
Status	**Open**		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Storage related to a contract is only persistent between transactions if it is inserted in the storage trie.

The counter `NEXT_TRIE_OBJECT_INDEX`, in `near_bindgen::Collections`, is used to create different storage keys, for each `Set` and `Map` object. It is defined as a `pub(crate) static mut`, however as the counter is not persistent between function calls, `Collections` objects that are created in separate transactions will have overlapping prefixes and therefore storage keys. Overlapping storage keys will cause all stateful functions between objects to collide.

This issue will also appear in any third party imported crates which use some form of global storage. Global storage can appear in multiple forms such as `pub(crate) static mut` and `lazy_static!static ref ARRAY: Mutex<Vec<u8>»`.

For a proof-of-concept, see test case `tests/collections-static`.

Recommendations

All variables should be stored persistently on the state trie if they are to be used in separate transactions.

NEAR-04	Overwriting Contract State Storage		
Asset	`near-bindgen`		
Status	**Open**		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The contract state is inserted into the storage trie with the key `B"STATE"`. A short simple key makes accidental or deliberate collision highly probable.

A storage write collision will cause the contract to be unusable if the overwritten data is not deserialisable. If the contract state is serialisable, there will be unchecked changes to the contract state which will lead to undefined behaviour, depending on the contract.

For a proof-of-concept, see test case `tests/state-overwrite`.

Recommendations

We recommend including an additional API function within `nearcore` for writing to contract state and enforcing the contract state to only be written to via the `state_write()` API function as opposed to `storage_write()`.

NEAR-05	Insecure Storage of Sensitive Information		
Asset	NEAR Wallet		
Status	**Open**		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The NEAR wallet (available at https://wallet.nearprotocol.com) allows users to interact with the NEAR Blockchain from the comfort of their favorite web browser, enabling them to:

- Create accounts;
- Send NEAR tokens;
- Receive tokens.

The private key associated with the user account is stored in **cleartext** in the browser local storage. The unencrypted private key is therefore available on the file system (with Firefox on Arch linux, it lives at the following location: `/.mozilla/firefox/xxxx.default/webappsstore.sqlite`).

Information saved in a browser's local storage is cached locally for an indefinite period of time. Any person having access to the workstation of a NEAR wallet user can read this file and retrieve the wallet's private key, and steal the victim's funds.

Information stored in the browser's local storage may be recovered if the browser and/or underlying operating system are compromised. Similarly, users may login to the application on untrusted devices such as shared computers and expose their information and private keys to other users of the device/workstation.

Recommendations

Make sure unencrypted private keys are not kept in the browser's local storage. Consider forcing users to encrypt their private keys with a strong password (using the AES256 cipher).

NEAR-06	Block Production Speedup		
Asset	`nearcore`		
Status	**Open**		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

There is no restriction on the minimum amount of time that must pass before processing and validating the next block. Honest clients have a minimum delay before they will be able to produce the next block; however, when validating a block, this minimum delay is not enforced. There is also no system to rebalance the block production to tend towards expected block production times.

Malicious users may deliberately increase their block production times which will increase block production of the entire network. They may also reduce the time the next block producers spend waiting for the minimum delay by purposefully setting their time in the past (must still be greater than the previous block).

The major impact is that the block production would be permanently brought forward. The amount the malicious user could bring blocks forward is directly proportional to their percentage of the overall stake (i.e. percentage of blocks they produce).

Recommendation

The attack vector described above is known by the development team and will be taken into consideration in the new fork choice design.

NEAR-07	Initialisation Function Incorrectly Set		
Asset	`near-bindgen`		
Status	**Open**		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

When an initialisation function is specified in the `#[near_bindgen(init => func)]` macro but is not defined in the related `impl`, a contract will successfully compile and deploy.

However, state deserialisation will panic which impacts any function taking `self` as an input.

For a proof-of-concept, see test case `tests/no-init`.

Recommendations

We recommend adding a compile time check to ensure that the function exists if it is declared as an initialisation function.

NEAR-08	Collisions within `Collections`		
Asset	`near-bindgen`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

`Collections` objects, `Set` and `Map`, are given a storage key `prefix` (`u64`) concatenated with the `Set` / `Map` key (any `BorshDeserialize/BorshSerialize` value). The prefixes start at zero and increment by one.

Due to the simplicity of the key, there is a substantial probability that programmers will unknowingly write contracts that have a storage key collision. Furthermore, the low entropy increases the ability of malicious users to get storage collisions, again depending on contract designs.

Additionally, if a value is written with the same length prefix within the iterator range, the iterator will no longer function correctly.

For a proof-of-concept, see test case `tests/iterator-overlap`.

Recommendations

We recommend hardening `prefix` to reduce the probability of collisions.

NEAR-09	Overexpansion of Contract State		
Asset	`near-bindgen`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

NEAR smart contracts are limited in the amount of gas they may consume by both `prepaid_gas` and `max_gas_burnt`. Different operations have a different proportion of gas burnt.

As a result it is possible to increase the size of a contract state without breaching the `max_gas_burnt` amount.

A future transaction which requires deserialising the contract state will then exceed the `max_gas_burnt` limit. The contract state functions would then be rendered unusable.

For a proof-of-concept, see test case `tests/state-overwrite`.

Recommendations

We recommend reconsidering the gas consumption / burn amount for transactions to prevent locking of contract states.

NEAR-10	Accuracy of Reward Calculation Variables		
Asset	`nearcore`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The following variables are used to calculate the amount of rewards for validators:

- `epoch_length`
- `max_inflation`
- `num_block_per_year`
- `total_supply`

`epoch_length` is set in the config, however it does not reflect the actual epoch length which may vary depending on when finalisation occurs.

`max_inflation` is compounding total supply per epoch and thus the Annual Percentage Yield (APY) is

$APY = (1 + \frac{0.05}{epochsperyear})^{epochsperyear}$.

When epoch length is set to 5 minutes, the $APY = 5.13\%$ rather than 5%.

`num_block_per_year` also varies significantly based upon the production speed of staking clients.

Recommendations

We recommend changing `epoch_length` to reflect the actual epoch length and disclosing the APY is 5.13%

NEAR-11	API Does not Enforce Strict Transport Security (HSTS)		
Asset	NEAR Wallet and Explorer		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The application servers hosting the NEAR wallet and the NEAR explorer fail to include the *HSTS* response header.

HTTP Strict Transport Security, or *HSTS*, is a protection mechanism that instructs browsers to solely communicate over HTTP**S** connections, and also used to prevent users from accepting invalid certificates.

An attacker able to modify a legitimate user's network traffic could bypass the extension's use of SSL/TLS encryption, and use it as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP, so that if a targeted user's browser does not attempt to use an encrypted connection.

The lack of HSTS can also be exploited by presenting invalid certificates to users that may accept it and by performing Man-in-the-Middle attacks using tools and techniques such as `SSLStrip`.

Recommendations

Enable HTTP Strict Transport Security (HSTS) by adding a HTTP response header labelled `Strict-Transport-Security`, with a value of `'max-age=expireTime'`, where `expireTime` is the time in seconds that browsers should remember that the site should only be accessed using HTTPS.

NEAR-12	API Does not Enforce Strict Transport Security (HSTS)		
Asset	NEAR Wallet and Explorer		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The application servers hosting the NEAR wallet and the NEAR explorer fail to include the *HSTS* response header.

HTTP Strict Transport Security, or *HSTS*, is a protection mechanism that instructs browsers to solely communicate over HTTP**S** connections, and also used to prevent users from accepting invalid certificates.

An attacker able to modify a legitimate user's network traffic could bypass the extension's use of SSL/TLS encryption, and use it as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP, so that if a targeted user's browser does not attempt to use an encrypted connection.

The lack of HSTS can also be exploited by presenting invalid certificates to users that may accept it and by performing Man-in-the-Middle attacks using tools and techniques such as `SSLStrip`.

Recommendations

Enable HTTP Strict Transport Security (HSTS) by adding a HTTP response header labelled `Strict-Transport-Security`, with a value of `'max-age=expireTime'`, where `expireTime` is the time in seconds that browsers should remember that the site should only be accessed using HTTPS.

NEAR-13	Weak TLS Protocols and Ciphers Supported		
Asset	NEAR Explorer and RPC API		
Status	Open		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The host serving the NEAR RPC API (`https://rpc.nearprotocol.com`) allows connections using insecure protocols and ciphers that are affected by significant flaws which have been publicly disclosed.

The following supported insecure protocols/ciphers have been identified:

- `TLSv1.0` and `TLSv1.1`
- Less than 128 bits of key length (`DES-CBC3` and `ECDHE-RSA-DES-CBC3` , both effectively providing 112 bits of entropy)

Additionally, the NEAR Blockchain explorer (`https://explorer.nearprotocol.com`) also supports ciphers with less than 128 bits (same ones as above).

Transport Layer Security (TLS) version 1.2 is the ITEF standard cryptographic protocol for secure communications. TLS provides authentication, confidentiality and data integrity between two communicating parties. SSLv3 and TLS v1.0 are superseded and suffer from publicly disclosed flaws and vulnerabilities that negate the advantages of using secure commination protocols.

Furthermore, ciphers less than 128 bits are also considered cryptographically insecure.

Exploitation of cryptographic protocol weaknesses is a complex attack that requires detailed knowledge of how the protocol operates and sufficient skills to perform an attack against users. Cipher based attacks often take considerable time and effort to perform, although are becoming easier over time with further research and increasing computing power. To attempt exploitation, an attacker must be in an adequate network position to intercept traffic between the user and the host.

Additionally, the server at `https://34.94.250.216` (hosting a copy of NEAR Studio) uses an expired SSL certificates (expired on Thursday, 5 September 2019).

Recommendations

We recommend:

- Using Transport Layer Security version 1.2 or 1.3 (`TLSv1.2` or `TLSv1.3`) for all communications and disabling the use of all legacy versions of SSL/TLS;
- Using only cipher suites that offer key lengths in excess of 128 bits.

It is generally recommended to use ciphers which conform to the following standards:

- Key exchange algorithm based on Diffie Hellman (DH) or Elliptical Curve Diffie Hellman (ECDH);
- A bulk encryption algorithm of AES_GCM that is at least 128 bits of entropy;
- A Message Authentication Code (MAC) and Pseudo-Random Function (PRF) of SHA-2;
- Supports secure renegotiation and uses Perfect Forward Secrecy.

If legacy TLS protocols are required for compatibility, use cipher suites that conform as close as possible to the above standards with the additional considerations of:

- Not using a cipher suite that uses RC4, 3DES, MD5 or IDEA;
- Not using cipher suites that use a bulk encryption algorithm of less than 128 bits;
- Enabling Transport Layer Security (TLS) Renegotiation Indication Extension (RFC-5746)

Also, make sure you renew all SSL certificates in use within the GCP infrastructure.

NEAR-14	Insufficient Protection Against Clickjacking		
Asset	NEAR Wallet		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The NEAR web wallet allows its page to be loaded within HTML frames, leaving it vulnerable to Clickjacking (also known as UI redressing) and Cross Frame Scripting (XFS). Clickjacking and XFS both involve tricking a user into performing undesired actions in the vulnerable application or revealing sensitive information.

The attack is carried out by utilising controlled *iFrames* to layer or conceal links, actions or scripts within a third party malicious page, inducing users to perform mouse clicks or keystrokes to unwittingly carry out actions.

Loading the application within a controlled frame can be used to steal user credentials by embedding JavaScript key logging code to capture keystrokes made to a login page that is loaded within an iFrame. This attack requires a user to be lured into visiting a malicious URL. Modern browsers are able to detect when frames are loading content across domains and will present warnings to a user.

Recommendations

Include the `X-Frame-Options` header in all server responses, which instructs the browsers to stop content from being framed. Use the *`DENY`* option to prevent all framing, or *`SAMEORIGIN`* to only allow framing by the origin domain if required for application functionality.

Legacy browsers may not support the `X-Frame-Options` header and in these cases require JavaScript based protection against clickjacking. Note that JavaScript based solutions cannot be relied upon as it has been proven that JavaScript based frame busting code can be bypassed and cannot be considered a security protection mechanism.

Information about Clickjacking and JavaScript based protection can be found at: https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

NEAR-15	Receiver Argument Type
Asset	`near-bindgen`
Status	Open
Rating	Informational

Description

A `FnArg::Receiver` function argument is a function argument which contains `self`. In comparison a `FnArg::Typed` argument which has a name and a type.

However, if `self` is written with a type i.e. `self: Box<Self>` then it is considered a `FnArg::Typed` rather than `FnArg::Receiver`.

Testing revealed that there will still be a compile time error when compiling the contract and thus no vulnerability actually arises from this behaviour.

For a proof-of-concept, see test cases `tests/fnargs-type` and `tests/fnargs-type-boxed`.

Recommendations

Be aware of the categorising of typed `self` arguments.

NEAR-16	Finalisation Fork
Asset	`nearcore`
Status	**Open**
Rating	Informational

Description

A user or conglomerate who has more than the mean stake are able to create finalisations on two forks. This is opposed to most finalisation methods which require 50% of total stake (excluding mass validator ejection). Consider the following:

1. Say a malicious user has percentage stake e and the mean attestations per block is x;
2. Starting at $block(i)$ honest producers will continue this chain for some amount, let say a, of blocks until $block(i)$ is 1 vote from finalisation;
3. The malicious users will produce a block forking from $block(i)$, and will immediately use all its votes to attest to this forked block. They must then produce one more block on top of the forked block which will then have a higher weight and become the new head for honest validators;
4. As this is the heaviest chain it will eventually get finalised by the honest validators and a proportion of the malicious users stake. At which point the malicious use produces one block on the original chain finalising it.

Recommendation

The attack vector described above is known by the development team and will be taken into consideration in the new fork choice design.

NEAR-17	General Design Comments
Asset	`nearcore`
Status	**Open**
Rating	Informational

Description

This section is used to make general high level comments about the current NEAR Blockchain design.

- An account can be created with or without a key. An account with a key is required to be able to deploy a contract to it. An individual who owns a key to an account that has a contract is able to delete the account (and therefore contract) at anytime, sending the related NEAR balance to a specified address. This encourages exit scams as malicious users may drain the contract balance at any time. However, users may delete their own key after deployment thereby making the contract immutable.
- Similarly a user with the key to an account can redeploy the new contract bytecode at any point without impacting the contract state trie.
- When a user deletes an account any other user may then create a new account using the same `accountId`.
- `Challenges` can be included in a block by validators, however these object don't seem to be used to compute the block hash. *Note: the testing team could not assess the impact of this potential issue within the allocated timeframe of this assessment.*

Recommendations

Consider allowing the deployment of contract directly to a new account which does not have a key and is unable to generate a key.

NEAR-18	Denial-of-Service Protection not in Use (Cloud Armor)
Asset	NEAR GCP Infrastructure
Status	Open
Rating	Informational

Description

Distributed denial of service (DDoS) attacks can render an entire infrastructure unusable. NEAR's GCP infrastructure could be targeted by malicious actors, for example to prevent users from accessing bootnodes. In a DDoS attack, the incoming traffic usually originates from multiple different sources, making it impossible to mitigate the attack by simply blacklisting a single IP source.

Google offers a new service named *Cloud Armor* which delivers defense at scale against infrastructure and application DDoS attacks by leveraging Google's global security resources (including a global, software-defined HTTPS load balancer).

Please note that the testing team did not perform a DDoS attack against the NEAR GCP Infrastructure as part of this assessment.

Recommendations

Consider enabling Google's *Cloud Armor* on the NEAR GCP infrastructure to mitigate potential future denial of service attacks (pricing for this new service can be found here).

NEAR-19	Google Cloud Access Transparency Disabled
Asset	NEAR GCP Infrastructure
Status	**Open**
Rating	Informational

Description

Google's *Access Transparency* allows GCP customers such as NEAR Protocol to have a consolidated view over Google's administrative operations, by keeping logs of actions performed by Google staff when accessing NEAR data.

This allows customers to make sure that Google is only accessing data for valid business reasons (e.g. legitimate maintenance tasks).

This service is not currently used in the NEAR GCP infrastructure.

Recommendations

Consider enabling Google's *Access Transparency* on the NEAR GCP infrastructure to obtain access to logs produced when Google administrators access NEAR's GCP environment.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

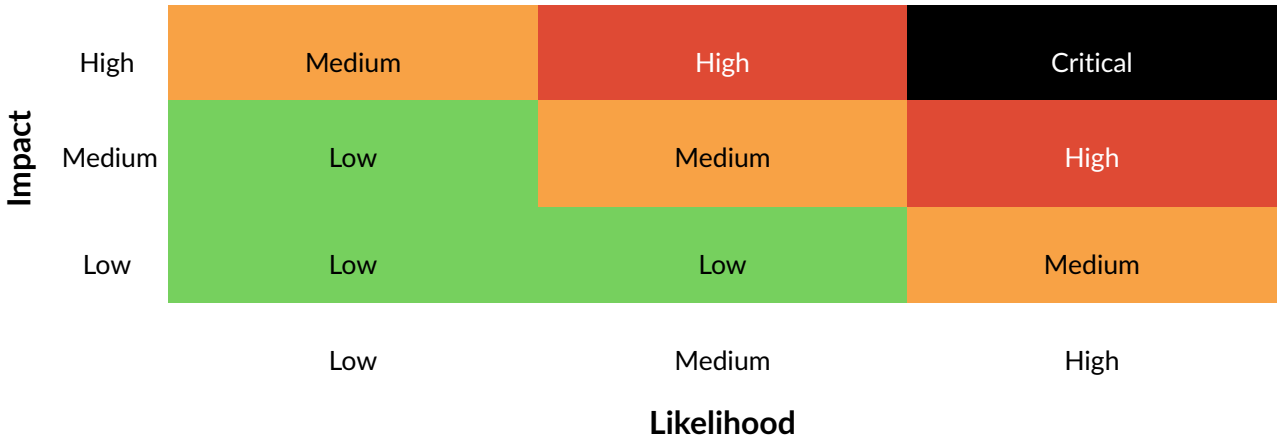

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.