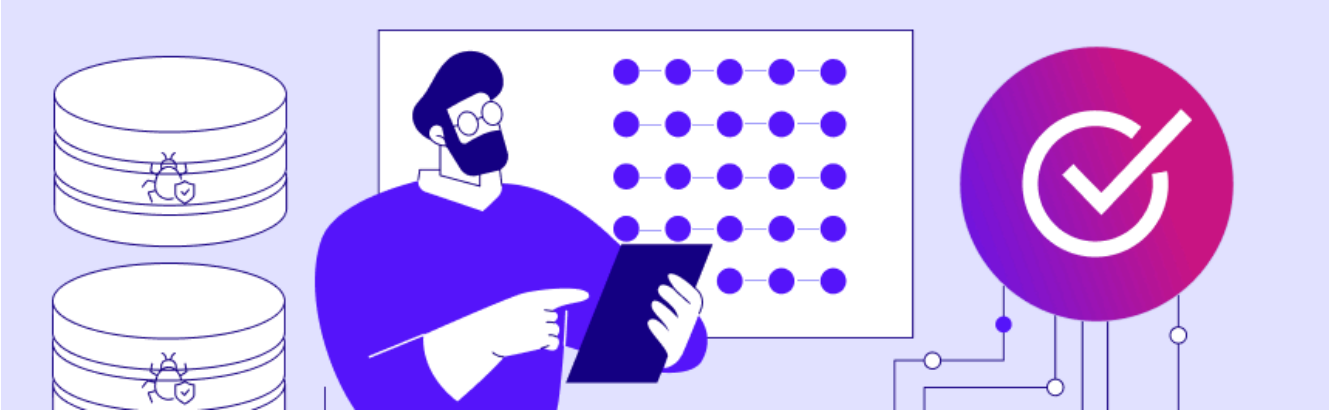
Uncovering EtherHiding: Malware Hidden in Blockchain

cymulate.com/blog/simulating-etherhiding-blockchain-as-a-malware

Ruben Enkaoua November 20, 2025



Executive summary

Following the discovery of the North Korea-based EtherHiding attack technique headlining a recent rise of blockchain abuse for malware delivery, Cymulate Research Labs analyzed and simulated the technique. EtherHiding embeds malicious payloads within the blockchain to persist and distribute malware without requiring attackers to maintain dedicated infrastructure.

By leveraging decentralized solutions, attackers can distribute malware, gain persistence and challenge censorship, since takedown is almost impossible in Web3. The risk is meaningful as payloads remain accessible, bypassing traditional network or AV controls, and supporting covert C2 or staged payload assembly that current SOC capabilities may struggle to detect.

To mitigate these threats, we recommend expanding monitoring to include blockchain/RPC retrieval activities, integrating blockchain indicators into threat intelligence and implementing RPC whitelisting to help identify and contain operations involving Web3 payload retrievals.

Our research and analysis of EtherHiding led us to build and release new attack scenarios for production-safe, live data testing that runs hidden payload stored in blockchain.

The simulation of web3 threats is not new to Cymulate. Cymulate assessment templates, threat feed and attack scenario workbench already include live-date testing for threats like crypto mining and crypto theft. The new attack scenarios simulating EtherHiding provide expanded coverage of innovative web3 threats not associated with crypto and apply to the broader attack tactics and techniques of C2, persistence and remote code execution.

What is EtherHiding? Why does it matter?

A blockchain is a distributed ledger with growing lists of records (blocks) securely linked together via cryptographic hashes. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data, forming a chain, with each additional block linking to the ones before it.

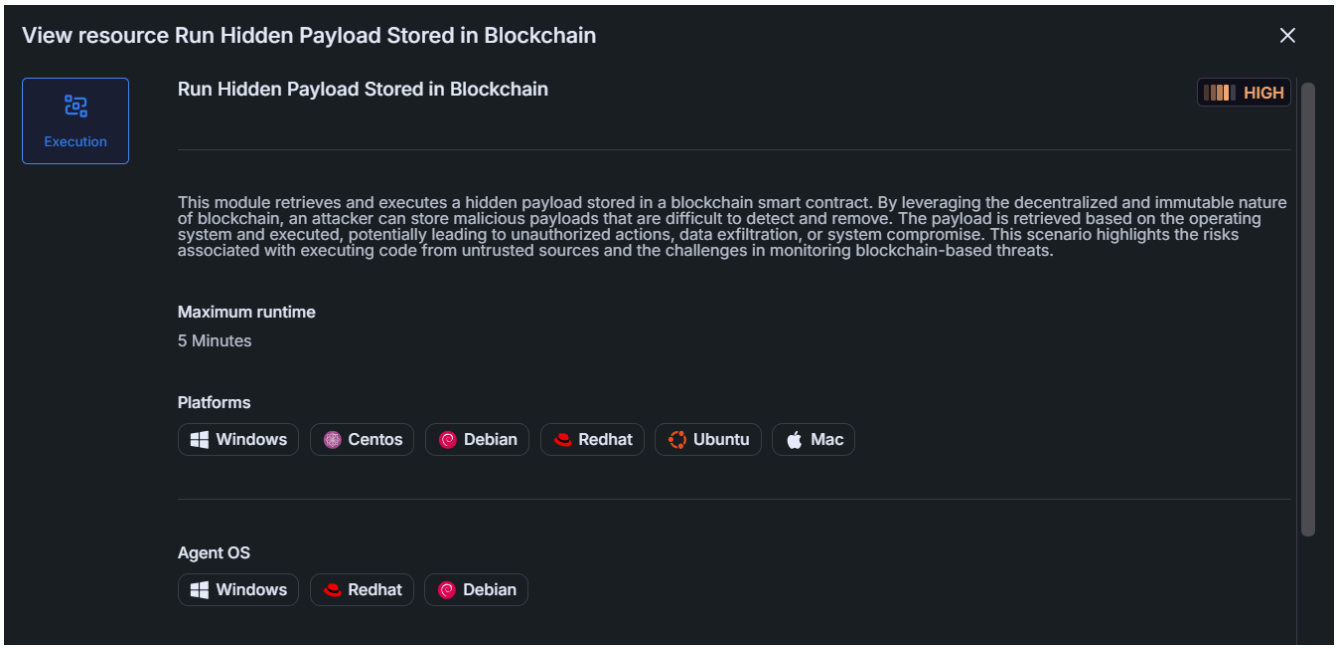
Consequently, blockchain transactions are resistant to alteration because, once recorded, the data in any given block cannot be changed retroactively without altering all subsequent blocks and obtaining network consensus to accept these changes.

According to a recent Google report, nation-state threat actors from North Korea were observed using EtherHiding to store and retrieve malicious payloads within smart contracts on a public blockchain like BNB Smart Chain or Ethereum.

This approach essentially turns the blockchain into a decentralized and highly resilient command-and-control (C2) server. Also, the technique was observed to be used to deploy malicious JavaScript to manipulate users into executing malicious code.

To understand the attack process, we built an attack scenario for Cymulate customers to simulate the new technique against their security controls like endpoint detection and response (EDR) and security information and event management (SIEM). The new Cymulate attack scenario extracts and runs a concealed payload embedded within a blockchain smart contract. By abusing the blockchain's decentralized and tamper-resistant characteristics, attackers can host malicious payloads that are challenging to detect or eliminate.

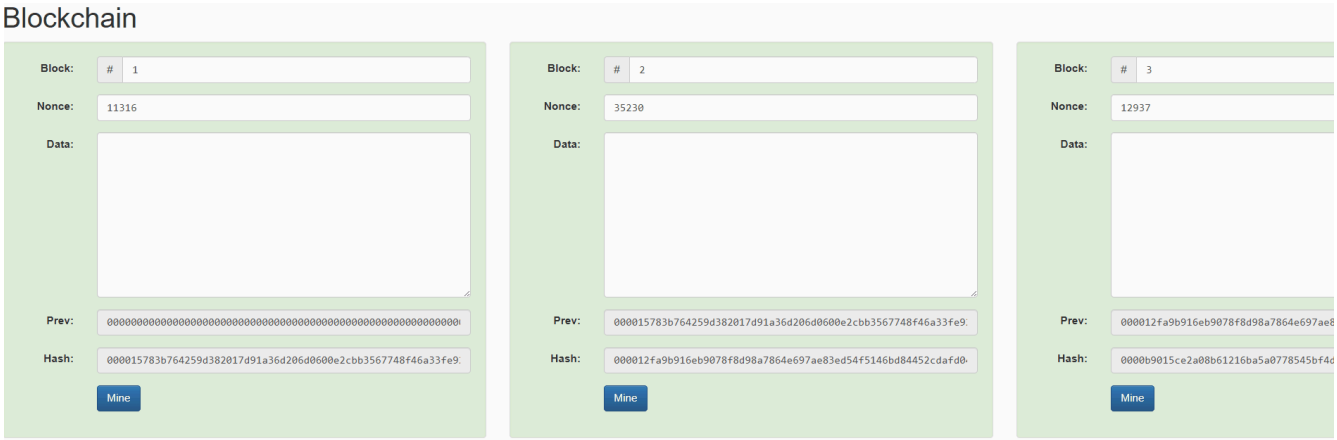
The payload is retrieved according to the target operating system and then executed, which may result in unauthorized actions, data theft, execution or persistence. This example underscores the dangers of executing code from decentralized sources and the complexities involved in detecting and mitigating blockchain-based cyber threats.



EtherHiding attack details

The “blockchain” is in fact composed of blocks of data, stored in consecutive groups. Transaction data must be added to a block to be validated. These blocks form a chain, assuming that each block cryptographically references its parent. To change a block, all subsequent blocks must change, which require the consensus of the entire network.

We can understand how it works from [this simulation](#) by Anders Brownworth.



By changing block #2, all blocks since then are invalidated. Ethereum for example uses a proof-of-stake based consensus mechanism, so changing block #2 would require recreating a new chain from this block and getting validators to sign new blocks to replace that history.

Assuming the attacker wouldn’t get the consensus of at least 66% of the validators, the alternate history will not be finalized and therefore will be rejected.

Programs can also run on Ethereum blockchain. These are called “smart contracts.” It is a collection of code and functions that reside at a specific address.

The following Solidity smart contract stores on the testnet the string Ruben:

NameStorage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract NameStorage {
    bytes private name;

    constructor(bytes memory _name) {
        name = _name;
    }

    function getName() public view returns (bytes memory) {
        return name;
    }
}
```

Deploy.s.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "forge-std/Script.sol";
import "../src/NameStorage.sol";

contract DeployNameStorage is Script {
    function run() external {

        uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");

        vm.startBroadcast(deployerPrivateKey);

        bytes memory myName = bytes("Ruben");
        NameStorage nameStorage = new NameStorage(myName);

        vm.stopBroadcast();
```

```
        console.log("Contract deployed at:", address(nameStorage));
    }
}
```

By running the following script, the name can be retrieved:

```
from web3 import Web3

RPC_URL = "https://ethereum-sepolia.publicnode.com"
CONTRACT_ADDRESS = "0xaE3Ba9d7C1374234b29525AE1EA07Ea93D30F5eD"

CONTRACT_ABI = [
    {
        "inputs": [],
        "name": "getName",
        "outputs": [{"internalType": "bytes", "name": "", "type": "bytes"}],
        "stateMutability": "view",
        "type": "function",
    }
]

def main():
    print("[+] Connecting to Ethereum node ...")
    w3 = Web3(Web3.HTTPProvider(RPC_URL))
    if not w3.is_connected():
        raise SystemExit("[!] Failed to connect")

    print("[+] Preparing contract handle ...")
    contract = w3.eth.contract(
        address=Web3.to_checksum_address(CONTRACT_ADDRESS),
        abi=CONTRACT_ABI,
    )

    print("[+] Retrieving bytes from contract ...")
    data: bytes = contract.functions.getName().call()

    hex_list = ",".join(f"0x{b:02x}" for b in data)
    print(f"[+] Bytes as hex: {hex_list}")
    print(f"[+] Byte length: {len(data)}")

    with open("name.bin", "wb") as f:
        f.write(data)
    print("[+] Saved to name.bin")

if __name__ == "__main__":
    main()
```

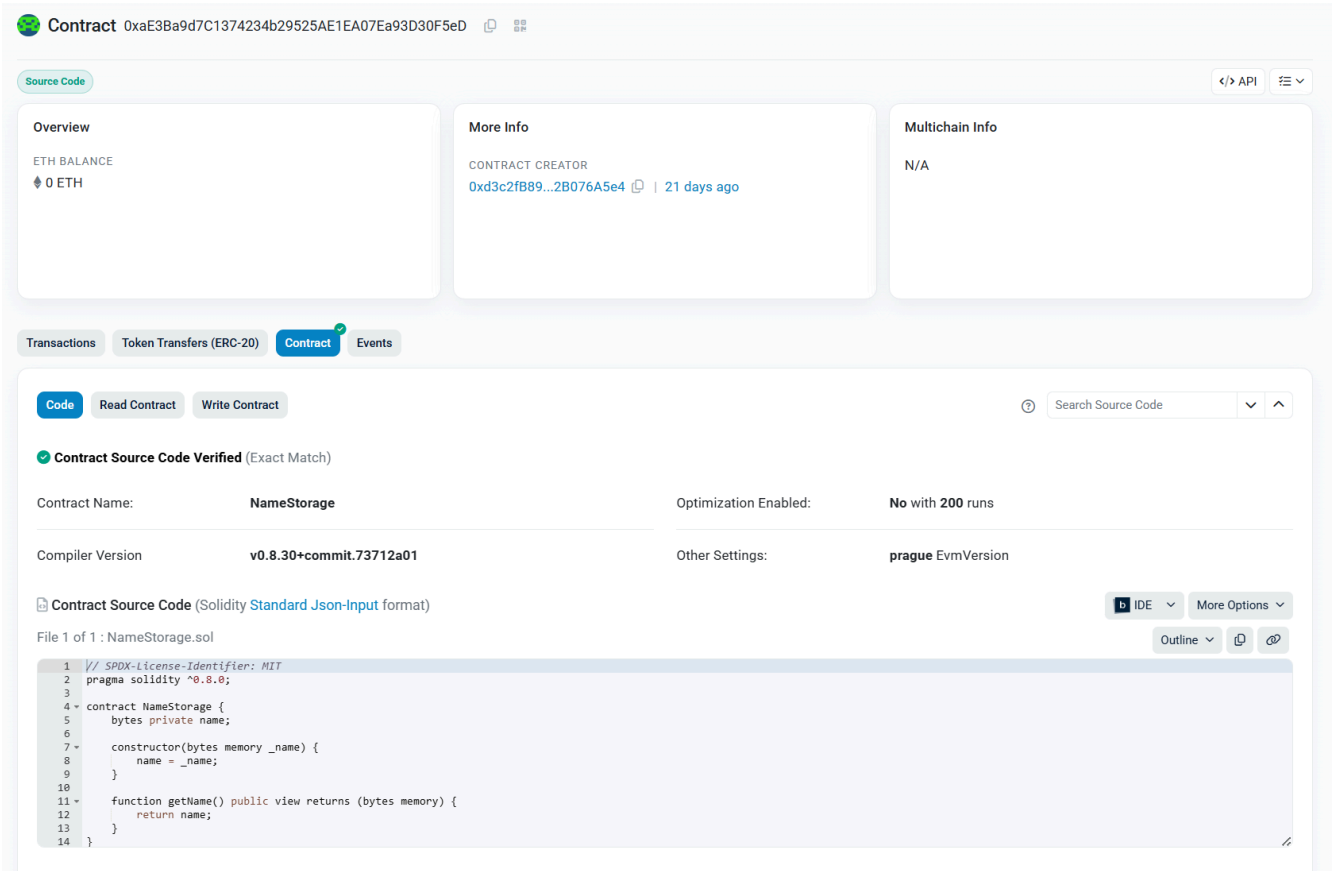
Output:

```
┌──(kali㉿kali)-[~/lab/web3/etherHiding]
└─$ python3 test.py
[+] Connecting to Ethereum node ...
[+] Preparing contract handle ...
[+] Retrieving bytes from contract ...
[+] Bytes as hex: 0x52,0x75,0x62,0x65,0x6e
[+] Byte length: 5
[+] Saved to name.bin

┌──(kali㉿kali)-[~/lab/web3/etherHiding]
└─$ cat name.bin
Ruben
```

Since the contract's bytecode is stored on-chain, anyone can read it, disassemble it and decompile it.

Our contract,
deployed at the address 0xAE3BA9D7C1374234B29525AE1EA07EA93D30F5ED, can be accessed from etherscan.io:

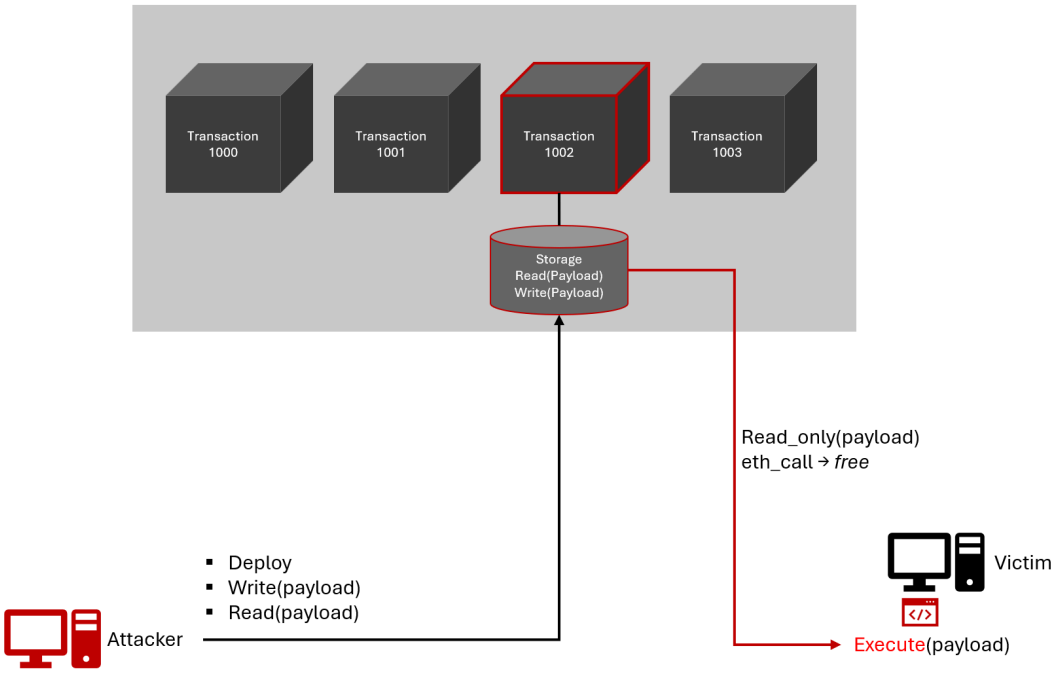


Assuming this, payloads can also be stored there and never be deleted.

To simulate a malicious payload, we created an execution testing the procedure by retrieving Linux or a Windows command depending on the victim Operating System.

```
┌──(kali㉿kali)-[~/lab/web3/etherHiding]
└─$ python3 test-stage.py
[+] Connecting to Ethereum node ...
[+] Detected OS: Linux
[+] Will call: getLos()
[+] Preparing contract handle ...
[+] Retrieving bytes from contract ...
[+] Executing Payload
Linux kali 6.16.8+kali-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.16.8-1kali1 (2025-09-24) x86_64 GNU/Linux
SUCCESS-ETHERHIDING-C2
```

The procedure would look like the following:



The read function, called by the malware, uses an RPC method (eth_call) and doesn't create a transaction.

Simulate EtherHiding with Cymulate today

EtherHiding demonstrates how adversaries are constantly evolving, leveraging new techniques and technologies to gain access, weaponize and maintain persistence over infected systems. Its decentralized approach opens new challenges and perspectives, shifting how we think about threat surfaces and defensive strategies.

Mitigating Etherhiding demands stronger client-side protection and more broadly, deeper visibility into Web3 interactions. This technique serves as a reminder that innovation in decentralization must be matched with innovation in security.

Cymulate customers can find the new attack simulation under the name “Run Hidden Payload Stored in Blockchain.”

Discover how you can prepare your defenses against the latest threats. See the Cymulate platform in action and sign up for a demo today.

Table of Contents

Cymulate Exposure Validation makes advanced security testing fast and easy. When it comes to building custom attack chains, it's all right in front of you in one place.

Mike Humbert, Cybersecurity Engineer

DARLING INGREDIENTS INC.

Learn More