Snipverse — The Web3 Immune System

A Developer■First Security Standard

Author: N.E.O. / Snipverse Network
Date: October 2025

> “We don’t prevent every hack. We make every user part of the immune response.”

The Problem

You ship a contract. You renounce ownership. You get audited. Still — one phishing link, one fake frontend, one rogue approval — and your users are drained before you wake up.

Why Current Tools Fail

Tool	Flaw
Etherscan labels	Too slow
Chainalysis	Closed, private
Exchanges	Weeks of silence
Community	Blind

The Snipverse Solution

Four on■chain primitives that turn users into sentinels:

Layer	Mechanism	Developer Benefit
1. On■chain Sentinel Layer	Signed watcher contracts	Users act as antibodies
2. Immutable Deployer Proof	`owner = 0x0` + registry badge	Code can’t be changed
3. Real■time Public Monitoring	Live tx feed + alerts	I see the attack as it starts
4. Instant Signal Hotline	Community → Etherscan tag → insurers	We react in minutes

This Is a Protocol, Not a Product

- Open standard
- Zero trust
- Community■owned
- Composable

> Think HTTPS padlock — but for Web3.

MVP in 6 Weeks

Week	Deliverable

1	`SnipRegistry.sol` + logo upload
2	`renounceAndLock()` + badge
3	Dune dashboard + Telegram bot
4	“Hotline” signed alert → Etherscan tag
5	Wallet UI integration
6	Launch with 5 real projects

Developer Quickstart

```
npm install @snipverse/shield

import { SnipShield } from "@snipverse/shield";

contract MyToken is ERC20, Ownable, SnipShield {
    constructor(address registry)
        ERC20("MyToken", "MTK")
        SnipShield(registry, address(this))
    {}

    function launch() external onlyOwner {
        renounceOwnership();
        _snipLockForever(); // Shield activated
    }
}
```

Appendix — Full Source Code

This appendix contains all referenced source files in full, formatted for line■by■line developer review.

SnipRegistry.sol (Solidity)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract SnipRegistry {
    struct Project {
        address deployer;
        bytes32 logoHash;
        uint256 lockedAt;
        string name;
        string website;
        bool exists;
    }

    mapping(address => Project) public projects;
    address[] public projectList;

    event Registered(address indexed contractAddr, address deployer, bytes32 logoHash, string name);
    event LogoUpdated(address indexed contractAddr, bytes32 newHash);

    function register(
        address contractAddr,
        bytes calldata logoSVG,
        string calldata name,
        string calldata website
    ) external {
        require(!projects[contractAddr].exists, "Already registered");
        bytes32 hash = keccak256(logoSVG);

        projects[contractAddr] = Project({
            deployer: msg.sender,
            logoHash: hash,
            lockedAt: 0,
            name: name,
            website: website,
            exists: true
        });

        projectList.push(contractAddr);
        emit Registered(contractAddr, msg.sender, hash, name);
    }

    function lockForever(address contractAddr) external {
        Project storage p = projects[contractAddr];
```

```
        require(p.exists && p.deployer == msg.sender && p.lockedAt == 0, "Cannot lock");
        p.lockedAt = block.timestamp;
    }

    function updateLogo(address contractAddr, bytes calldata logoSVG) external {
        Project storage p = projects[contractAddr];
        require(p.exists && p.deployer == msg.sender && p.lockedAt == 0, "Cannot update");
        p.logoHash = keccak256(logoSVG);
        emit LogoUpdated(contractAddr, p.logoHash);
    }
}
```

SnipShield.sol (Solidity, helper mixin)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

interface ISnipRegistry {
    function lockForever(address contractAddr) external;
    function projects(address)
        external
        view
        returns (
            address deployer,
            bytes32 logoHash,
            uint256 lockedAt,
            string memory name,
            string memory website,
            bool exists
        );
}

/// @title SnipShield
/// @notice Minimal helper mixin that calls into SnipRegistry to lock a target contract forever.
abstract contract SnipShield {
    address public immutable snipRegistry;
    address public immutable snipTarget;

    event SnipLocked(address indexed target, uint256 lockedAt);

    constructor(address registry, address target) {
        require(registry != address(0) && target != address(0), "Invalid address");
        snipRegistry = registry;
        snipTarget = target;
    }

    /// @dev Call this from your launch path after renouncing ownership.
    function _snipLockForever() internal {
        ISnipRegistry(snipRegistry).lockForever(snipTarget);
        (, , uint256 lockedAt, , , ) = ISnipRegistry(snipRegistry).projects(snipTarget);
```

```
        require(lockedAt > 0, "Lock failed");
        emit SnipLocked(snipTarget, lockedAt);
    }
}
```

SnipShield.tsx (React + wagmi)

```
// SnipShield.tsx
import { useContractRead } from 'wagmi';
import type { Address } from 'viem';

const REGISTRY: Address = '0x...';

export default function SnipShield({ contractAddr }: { contractAddr: Address }) {
  const { data: project } = useContractRead({
    address: REGISTRY,
    abi: snipRegistryABI,
    functionName: 'projects',
    args: [contractAddr],
    watch: true
  });

  if (!project || !project.exists) return <div>Not Snip-protected</div>;

  const isLocked = Number(project.lockedAt) > 0;
  const label = isLocked ? 'Locked' : 'Active';

  return (
    <div className="snip-shield" style={{ display: 'flex', gap: 8, alignItems: 'center' }}>
      <img src={`...`} width={20} height={20} alt="Snipverse Badge" />
      <span style={{ fontWeight: 600, color: isLocked ? '#22c55e' : '#eab308' }}>{label}</span>
    </div>
  );
}
```

Dune SQL — Live Monitoring Query

```
-- Dune SQL: High-value movements for registered projects in the last 5 minutes
SELECT block_time, tx_hash, "from", value / 1e18 AS eth
FROM ethereum.traces
WHERE contract_address IN (SELECT project FROM snipverse_projects)
  AND value > 1e18
  AND block_time > now() - interval '5 minutes';
```

telegram-bot.js (Node.js, polling)

```
// telegram-bot.js
// npm i node-telegram-bot-api axios
import TelegramBot from 'node-telegram-bot-api';
import axios from 'axios';

const BOT_TOKEN = process.env.TELEGRAM_TOKEN || 'TOKEN';
const CHAT_ID = process.env.CHAT_ID || '123456';
```

```
const DUNE_ENDPOINT = process.env.DUNE_ENDPOINT || 'https://api.dune.com/api/v1/query/.../results';

const bot = new TelegramBot(BOT_TOKEN, { polling: true });

async function fetchDuneResults() {
  const { data } = await axios.get(DUNE_ENDPOINT, {
    headers: { 'x-dune-api-key': process.env.DUNE_API_KEY || '' }
  });
  // adapt to your Dune response shape
  return data.result?.rows || [];
}

setInterval(async () => {
  try {
    const rows = await fetchDuneResults();
    rows.forEach((row) => {
      const msg = `*Snipverse Alert*\nValue: ${row.eth} ETH\n[Tx](https://etherscan.io/tx/${row.tx_hash})`;
      bot.sendMessage(CHAT_ID, msg, { parse_mode: 'Markdown' });
    });
  } catch (e) {
    console.error('Polling error', e);
  }
}, 60_000);
```

scripts/deploy.js (Hardhat)

```
// scripts/deploy.js
// npx hardhat run scripts/deploy.js --network <network>
const hre = require('hardhat');

async function main() {
  const SnipRegistry = await hre.ethers.getContractFactory('SnipRegistry');
  const registry = await SnipRegistry.deploy();
  await registry.deployed();
  console.log('SnipRegistry deployed to:', registry.address);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

Example: MyToken.sol (using SnipShield)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import './SnipShield.sol';
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';
import '@openzeppelin/contracts/access/Ownable.sol';
contract MyToken is ERC20, Ownable, SnipShield {
```

```
    constructor(address registry)
        ERC20('MyToken', 'MTK')
        SnipShield(registry, address(this))
    {}

    function launch() external onlyOwner {
        renounceOwnership();
        _snipLockForever(); // Shield activated
    }
}
```

Notes & Security Considerations

• Review registry permissions and consider adding EIP■712 signed requests for delegated locking. • Ensure your deployment process captures the registry address immutably. • Consider chain■agnostic IDs for cross■chain registries. • Production deployments should add event indexing and circuit breaker patterns where appropriate.