Web3 Security Roadmap

by Mono Audit

ver. 1.0

Web3 Security Roadmap

The Web3 Protocol Security Roadmap is a strategic document that helps project teams plan and execute security-related tasks throughout the protocol lifecycle.

It functions both as a practical guide for internal use and as a communication tool to inform the community of the team's security efforts.

The Roadmap evolves with the product and reflects both planned and completed activities aimed at improving the security of the protocol.

Lifecycle Stages

The Roadmap covers all stages of the protocol's life, from ideation to post-launch maintenance.

It divides security-related activities into four stages: **Planning**, **Development**, **Pre-deployment**, and **Post-deployment**.

The **Planning** stage begins when the founders commit to launching the protocol. This is where the foundational work begins, even if coding has not yet begun.

The **Development** stage begins when engineers begin writing code, and can continue in cycles throughout the protocol's lifespan, especially with frequent updates and iterations.

The **Pre-deployment** stage occurs when the dev phase is complete but before launch. This is a risky period: teams are often under pressure to launch, but rushing it can have disastrous consequences. This stage should focus on verifying security readiness and addressing any critical issues.

The **Post-deployment** stage includes ongoing operations such as monitoring, updates and migrations, maintaining the long-term stability of the protocol.

Optionality in Roadmap Items

The roadmap uses three levels of importance.

Mandatory actions are non-negotiable for any security-focused protocol.

Nice-to-have actions may not apply in every case, but should be considered necessary when appropriate.

Optional actions may be skipped by team depending on context, although their implementation can add another layer of protection.

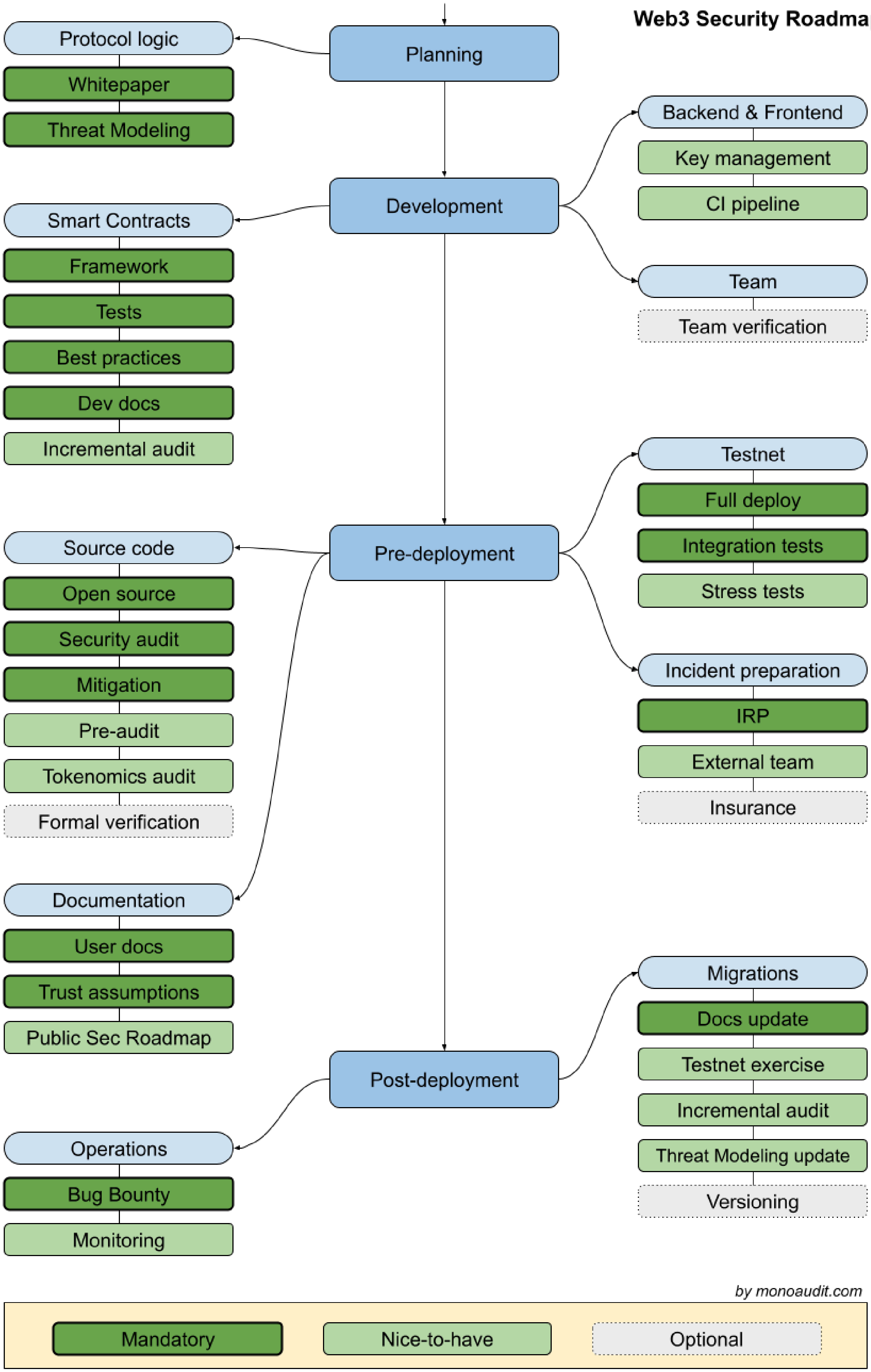
Roadmap Publication

Making the Web3 Security Roadmap public allows the team to build trust with users by clearly communicating their security strategy.

Early versions of The Roadmap will describe intent only. As development progresses, the points in the security plan will evolve from declarations to statements of fact.

As the protocol evolves, The Roadmap should be maintained as a living document with version control and a history of changes, allowing users and participants to easily track progress and understand the security posture of the protocol.

Web3 Security Roadmap



by monoaudit.com

Planning

Protocol Logic

Documentation

It is important to document the core logic of the protocol early in the planning phase.

Whether in the form of a whitepaper or interactive documentation, this helps team members align on implementation goals, serves as a key reference for auditors, and gives users a clear overview of the protocol's intended functionality.

This documentation should be publicly available and kept up to date.

Threat Modeling

Threat modeling should begin after the protocol architecture has been defined, but before coding begins.

This involves analyzing the flow of values and data through the protocol, mapping its dependencies, and identifying potential attack vectors.

The resulting document should describe the risks, their potential impacts, and mitigation strategies.

If the threat model reveals critical flaws, the protocol logic should be revised.

Publishing this information demonstrates the team's commitment to proactive security.

Development

Smart Contracts

Established Framework

Once development begins, choosing a modern, widely adopted smart contract framework becomes important.

This decision simplifies internal workflows and makes it easier for the community, auditors, and contributors to interact with the codebase.

Such frameworks typically offer robust ecosystems with tools, linters, and plugin integrations.

Automated tests

Testing is a fundamental requirement during the development phase.

Writing unit tests, integration tests, and where applicable fuzz tests ensures that the codebase is working reliably.

A well-integrated CI pipeline should automatically run these tests after each change, preventing broken code from being merged.

Transparency about test coverage and results increases trust with users and external stakeholders.

Best practices

Following established best practices in smart contract development and security helps teams avoid common pitfalls.

By incorporating these practices into development through frameworks and CI automation, teams lay a solid foundation for secure protocols.

Developers Documentation

Maintaining up-to-date developer documentation supports both internal knowledge transfer and external collaboration.

As team members come and go, documentation ensures continuity.

For external auditors, contributors, and researchers, good documentation reduces the learning curve and helps them engage more effectively with the code.

Incremental Security Audit

Traditional audits provide a snapshot of security at a specific point in time, but incremental audits follow the development process continuously.

These audits start with the first code commit and track vulnerabilities as the code evolves.

Each time new code is added, the auditor focuses only on the latest changes.

This approach shortens the feedback loop, helps developers fix issues faster, and reduces the workload of security reviewers.

Backend & Frontend

Hot wallet keys management

Security is not limited to smart contracts.

Any internal or external systems require careful handling, especially when dealing with hot wallet keys or administrative privileges.

Key leaks remain the leading cause of protocol breaches, so teams should rely on proven secret management solutions.

CI pipelines for Security

In addition to basic CI pipelines for testing, integrating tools that scan for vulnerabilities in dependencies can protect against supply chain attacks.

These tools help identify outdated or vulnerable packages during the build process and can be automated as part of CI/CD workflows.

Team

Team verification

The human factor must be taken into account.

Insider threats are real, especially in high-value protocols.

Teams should use auditing tools and role-based restrictions to minimize risk.

A team verification service will help identify suspicious individuals or restrict their actions.

Publicly disclosing team members and contributors can also build user trust.

Pre-deployment

Source code

Open source & Smart contracts verification

As the product nears launch, it's essential to make the source code open and verify smart contracts on-chain.

In Web3, closed source is more of a red flag than a security measure.

Attackers can still analyze the bytecode, while transparency encourages the community to contribute to security.

Pre-audit checklist

A pre-audit is a lightweight process designed to identify issues before a formal audit.

It identifies missing documentation, failed tests, and broken contracts early, saving time and money during the full audit phase.

Security audit

A full audit remains the cornerstone of Web3 security.

While it does not guarantee absolute security, a professional review significantly reduces the risk of serious flaws.

Explicit disclosure of unfixed vulnerabilities

Following an audit, any decision not to patch identified vulnerabilities should be explained publicly, including the rationale and any risk implications.

Economic Model Audit

As protocols become more complex, it becomes increasingly difficult to reason about economic logic.

Logic-focused audits are becoming increasingly important, especially to detect errors in interconnected systems or tokenomics.

These audits address issues that basic smart contract audits may miss.

Formal Verification

Formal verification, although resource-intensive, can provide mathematical certainty around critical parts of a protocol's logic.

It reduces human error and cognitive bias, making it a powerful tool when applied selectively to the most sensitive components.

Documentation

User documentation

User documentation doesn't just help end users.

For developers and auditors, this level of documentation fills the gap between technical implementation and actual functionality.

It supports the creation of accurate mental models, especially when the code is abstract or low-level.

Explicit disclosure of trust assumptions

Clearly stating the trust assumptions underlying your protocol is another sign of maturity.

From reliance on third-party contracts to multi-signatures and admin permissions, users need to know what is outside your direct control.

This includes internal infrastructure, wallets, or off-chain systems involved in the user journey.

Explicit disclosure of Security Roadmap and its status

Publishing the security roadmap itself, along with its status, ensures that users can verify the project's intent and track progress.

It should be presented clearly and include links to audits, dashboards, and other security-related materials.

Testnet

Full deploy

Deploying a full protocol version to a testnet provides an opportunity to rehearse deployments and test features in a safe environment.

It should use the same interfaces as the mainnet to simulate the real user experience.

Test: on-chain integrations; kill switch

Integration testing and incident response exercises can also be performed here.

Incentivized testnet real user stress-test

Testnets can also support marketing and feedback collection.

By incentivizing users to participate in testnet usage, teams can identify usability and performance issues under realistic load conditions while building community engagement.

Incident preparation

Incident response plan

Even if your protocol seems airtight, having an incident response plan is essential.

This plan should detail roles, communication processes, emergency shutdown procedures, coordination with legal and security experts, and other critical response tasks.

It should be reviewed regularly and used in practical exercises.

Blue team agreements

Partnering with an external security team for incident response ahead of time can save precious minutes when an attack happens.

Publicly disclosing this partnership shows that you take security seriously.

Loss insurance

Integration with decentralized insurance protocols can provide users with the ability to protect their funds.

These services allow users to share risk and create an additional safety net, which reflects positively on your project's commitment to user security.

Post-deployment

Operations

On-chain monitoring

Continuous monitoring of on-chain activity becomes a frontline defense.

Sudden anomalies should trigger alerts so your team can quickly respond and prevent damage.

A timely identified attack attempt can be stopped by activating an emergency protocol.

Bug bounty

Launching a public bug bounty program invites ethical hackers to test your protocol.

With clearly defined reporting channels and meaningful rewards, these programs attract attention and encourage disclosure instead of exploitation.

Migrations

Testnet migration exercise

Testnets also play a role after launch.

Testing migration scenarios and new deployments on testnets can prevent real bugs from happening in the first place.

It also reduces the pressure on developers during important updates.

Up-to-date documentation

Maintaining up-to-date technical and user documentation is part of responsible operations.

Any changes in logic, dependencies, or deployment process should be documented.

Incremental security audit

Every protocol update should go through an incremental security review.

Continuous auditing allows teams to respond to changes effectively without starting from scratch.

Threat Modeling Review

Major updates should trigger a new review of the protocol's threat model.

Even small changes in integration or dependencies can introduce new risks.

Updated models should be published for transparency.

Frontend and backend proper versioning

Applying sensible versioning practices for both frontend and backend code makes it easier to identify issues and revert to stable versions during incidents.

This approach can reduce confusion for users and prevent reputational damage during unexpected outages.