[Public]
Note: this document is incomplete and a draft to facilitate discussion toward a KEP.
Last updated: 8/29/2025

KEP: Community Security Patches for Out-of-Support Kubernetes Versions

Summary

This KEP proposes establishing a new repository `kubernetes/security-patches` to maintain security patches for Kubernetes minor versions that are no longer officially supported. This leverages the existing Security Response Committee (SRC) process and test infrastructure while making best-effort security patches available for out-of-support versions for an additional 12 months beyond the current support window of 14 months.

Motivation

Background

The Kubernetes project currently maintains an N-2 support policy, providing patch support for the three most recent minor releases for approximately one year each. However, real-world deployment patterns show that many organizations continue running versions that fall outside this support window.

Goals

- Extend the existing SRC responsibilities to include reviewing community-driven public security patches for versions that are out-of-support for less than 12 months
- Provide a mechanism for the community to maintain Kubernetes CVE security patches for recently out-of-support Kubernetes versions
- Maintain clear separation between officially supported and community-maintained patches
- Minimize additional CI/infrastructure burden on the project

Non-Goals

- Extending official project support beyond the current N-2 policy
- Pushing updates to out-of-support kubernetes branches
- Extending bug bounty or CVE issuance for out-of-support versions
- Providing feature backports or non-security bug fixes
- Creating an obligation for maintainers to support older versions indefinitely
- Replacing the need for timely upgrades to supported versions
- Maintain Kubernetes docs for out-of-support versions

Proposal

Repository Structure

New kubernetes repo structure:

```
kubernetes/security-patches/
├── README.md                    # Usage guidelines and disclaimers
├── v1.30/                       # Recently out-of-support version
│   ├── 0001_CVE-2025-XXXX.patch
│   ├── 0002_CVE-2025-YYYY.patch
── v1.29/                       # Older out-of-support version
│   ├── 0001_CVE-2025-XXXX.patch
└── tools/
    ├── apply-test-security-patches.sh
```

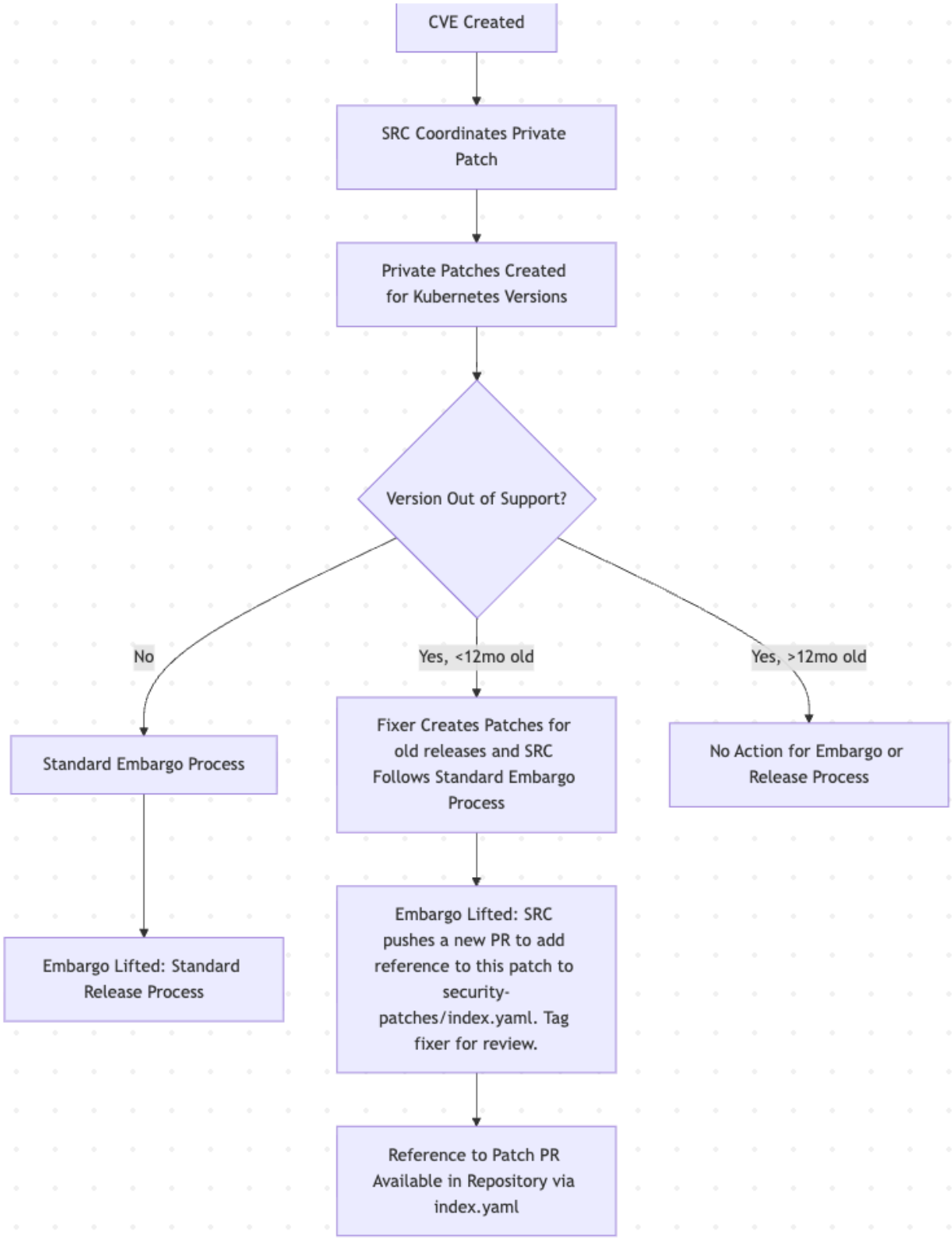
Enhanced SRC Process

Current SRC Process Extension

The Security Response Committee already coordinates security responses for supported Kubernetes versions. This proposal extends that process to include:

1. **12-Month Extended Coverage**: When a Kubernetes version goes out of official support, SRC continues to coordinate reviews of security patches for an additional 12 months
    a. Embargo CVE process will include SRC to encourage fixers to provide patches for EOL versions (3 versions and best effort) at the time they are creating private patches.
    b. When the CVE goes public, SRC can push a PR to the patch folder and tag the fixer for review.
2. **security-patches repo**: Security patches for out-of-support versions are stored in the `security-patches/` repository.

SRC Embargoed CVE Workflow Integration



```mermaid
graph TD
    A[CVE Created] --> B[SRC Coordinates Private Patch]
    B --> C[Private Patches Created for Kubernetes Versions]
    C --> D{Version Out of Support?}
    D -->|No| E[Standard Embargo Process]
     E --> F[Embargo Lifted: Standard Release Process]
    D -->|Yes, <12mo old| G[Fixer Creates Patches for old releases and SRC Follows Standard Embargo Process]
    G --> H[Embargo Lifted: SRC pushes a new PR to add reference to this patch to security-patches/index.yaml. Tag fixer for review.]
    H --> I[Reference to Patch PR Available in Repository via index.yaml]
    D -->|Yes, >12mo old| K[No Action for Embargo or Release Process]
```

Patch Management Without Extensive CI

Leveraging Existing Infrastructure

Instead of creating new CI pipelines, this proposal leverages existing Kubernetes infrastructure:

1. **Manual Testing Process**: CVE Fixer test patches against locally built Kubernetes versions
2. **PR Gate**: Minimal PR gate for these PRs made to the new repo by pulling from existing kubernetes release branch and temporarily apply all existing patches and the new patch for basic validation
3. **Community Validation**: Rely on Fixer and SRC review rather than automated CI

Testing Strategy for New Security Patches

Solution - Cumulative Patch Testing as a PR gate:

```bash
#!/bin/bash
# tools/apply-and-test-patch-on-release.sh

    local release_branch="$1"
    local new_patch="$2"
    local patches_dir="${release_branch}"
    local test_branch="test-security-patch-${release_branch}-$(date +%s)"

    echo "Starting security patch testing for Kubernetes release-${release_branch}"

    # 1. Setup clean git environment
    git fetch upstream
```

```
    git checkout "upstream/release-${release_branch}" -B "$test_branch"
    echo "Created test branch: $test_branch"

    # 2. Apply existing security patches in order
    if [[ -d "$patches_dir" ]]; then
        echo "Applying existing patches from $patches_dir..."
        for patch in $(find "$patches_dir" -name "*.patch" | sort); do
            if [[ "$patch" != "$new_patch" ]]; then
                echo "Applying existing patch: $(basename "$patch")"
                git apply --index "$patch"
                git commit -m "Apply security patch: $(basename "$patch")"
            fi
        done
    fi

    # 3. Apply the new patch being tested
    echo "Applying new patch: $(basename "$new_patch")"
    git apply "$new_patch" || {
        echo "Failed to apply new patch: $new_patch"
        exit 1
    }
    git commit -m "Applied new security patch: $(basename "$new_patch")"

    # 4. Build all to verify
    # Use dockerized build for consistency
    echo "Running build test with dockerized environment..."
     build/run.sh make all

    echo "Patch testing completed successfully for release-${release_branch}"
```

Patch Requirements and Format

Security Patch Criteria

Patches accepted into the security-patches repo must:

1. **Address documented Kubernetes CVEs**
2. **Be derived from official patches** already applied to supported versions
3. **Link to the official CVE issue** which includes attributions to original CVE and upstream patch
4. **Be tested against the target version** with evidence provided from local testing
5. **Be minimal and targeted** - no feature additions or extensive refactoring

Patch File Format

```patch
# security-patches/v1.30/0001_CVE-2025-XXXX.patch
#
# Security patch for CVE-2025-XXXX
# Original patch: https://github.com/kubernetes/kubernetes/commit/abc123
# Severity: HIGH (CVSS 8.1)
# Affected components: kube-apiserver, kubelet
#
# Testing: Validated against v1.30.14 on 2025-07-28
# CVE Fixer: @username
# SRC Reviewer: @src-member
#
diff --git a/pkg/apiserver/server.go b/pkg/apiserver/server.go
index 1234567..abcdefg 100644
--- a/pkg/apiserver/server.go
+++ b/pkg/apiserver/server.go
[patch content]
```

Governance Model

Security Response Committee Oversight

- **SRC maintains authority** over all security patch decisions
- **12-month commitment** for patches after versions go out of support
- **CVE Fixer** assist with patch creation and testing under SRC guidance

Review Process

1. [Embargo] **CVE Fixer** creates private patch file from upstream kubernetes release branch, runs tools/apply-and-test-patch-on-release.sh (see below) and tests new patch against target version locally
3. [Post Embargo] **SRC member** pushes patch to repo and tags CVE Fixer for review
4. **Community validation** CVE Fixer reviews.
    – multiple contributors **(Should we request at least one more local test other than the PR submitter?)**
5. **SRC approval** required before patch is committed to security-patches repo

Supported Versions

- **12-month extended coverage** for versions after they go out of official support

- **Clear documentation** that patches are community-maintained, not officially supported

Testing and Validation

CVE Fixer Testing Requirements

Since we're avoiding new CI infrastructure, testing relies on community effort:

```markdown
## Testing Checklist for Security Patch Fixer

### Required Testing Evidence:
- [ ] Patch applies cleanly on top of target out-of-support kubernetes release branch and all previous security patches for that version
- [ ] All affected components build successfully
- [ ] Basic functionality test of affected components
- [ ] Security vulnerability mitigation verified
- [ ] No obvious regressions in core functionality

### Testing Process:
1. Check out target kubernetes release branch (e.g., release-1.30)
2. Apply all existing patches from security-patches/v1.30/ in order
3. Apply the new security patch
4. Build all: `make WHAT="<component-list>"`
5. Run component-specific tests if available
6. Document results in patch header comments
```

Documentation and Usage

Clear Disclaimers

All documentation must prominently include:

```markdown
## IMPORTANT DISCLAIMERS

- **Community Maintained**: These patches are community-maintained and NOT officially supported
- **No Warranty**: Use at your own risk with thorough testing
- **Upgrade Recommended**: Upgrading to supported versions is strongly recommended
- **Limited Compatibility**: Patches may not work with vendor distributions
- **Time Limited**: Patches are only maintained for 12 months after version EOL
```

Implementation Plan

Phase 1: Process Integration (Month 1-2)
- Establish kubernetes/security-patches repo structure
- Document enhanced SRC process for out-of-support versions
- Create basic tooling for patch management
- Train SRC members on extended process

Phase 2: Initial Patches (Month 3-4)
- Identify recently out-of-support versions (e.g., 1.29, 1.30)
- Create security patches for recent CVEs affecting these versions
- Establish community testing process
- Document procedures and guidelines

Phase 3: Community Adoption (Month 5-6)
- Announce availability to kubernetes-dev community
- Gather feedback on process and patch quality
- Refine procedures based on experience
- Establish regular cadence for patch maintenance

Success Metrics

- Number of security patches maintained for out-of-support versions
- Community engagement in testing and validation
- Adoption of patches by organizations running older versions
- SRC feedback on process integration
- Reduction in security exposure for out-of-support deployments

Risks and Mitigations

Risk: Creating false expectations of long-term support
Mitigation: Prominent disclaimers, clear 12-month time limit, and consistent messaging about upgrading.

Risk: Poor quality patches due to limited testing
Mitigation: SRC oversight, community testing coordination, and conservative patch acceptance criteria.

Risk: Maintenance burden on SRC
Mitigation: Community volunteer network, limited 12-month commitment, and focus on Kubernetes CVEs only.

Risk: Confusion about support boundaries

Mitigation: Clear documentation, separate repos, and explicit disclaimers in all materials.

Alternatives Considered

A Folder in the Same Repository
A security-patches folder in`kubernetes/kubernetes` repository. Cons:
- All of the CI jobs that run on kubernete/kubernetes would be irrelevant to these files
- We'd likely want CI jobs that are unique to these patches (attempting to automatically apply the patch cleanly to the relevant kubernete/kubernetes release branch, etc)
- Periods when kubernete/kubernetes is frozen during release lifecycles would make it harder to commit changes to this folder.

Extended Official Support
Extending official support was rejected due to maintainer capacity constraints, added CI costs, and the goal of encouraging upgrades.

Implementation Details

Repository Creation
- **Commit access**: Limited to SRC members and approved maintainers
- **Branch protection**: Standard protections apply to main branch

Graduation Criteria

Alpha (Initial Implementation)
- kubernetes/security-patches repository established
- Enhanced SRC process documented and operational
- First 3 security patches created for 1-2 out-of-support versions
- Basic tooling and documentation available

Beta (Proven Process)
- 5+ patches across 2-3 out-of-support versions
- Community testing process established with regular CVE Fixers
- SRC feedback indicating sustainable process

Stable (Long-term Sustainability)
- Established 12-month lifecycle for patch maintenance

Open Questions
- what about vendor dependencies that need to be updated
    - This is for K8s code
- would we accept a patch to allow go updates? As potentially needed for library bumps
    - ideally, building with new go would “just work” with no k8s changes

    - if it didn’t, we could allow a patch that would make building with new go possible (ideally build-tagged so the build *works* with new go but doesn’t force it?)
- next step:
    - update proposal goals / non-goals, do another review pass
    - share with stakeholder: SRC, sig-security, sig-arch (FYI), sig-release (FYI)
    - decide where this KEP should go: sig-security, new SRC folder? (requires steering)