Open source risks: perception & mitigation

Petr Špaček, Victoria Risk
2024-05-23
pspacek@isc.org, vicky@isc.org
Experimental session

- Survey
- BIND 9 DNS server
  - as an example project
- Discussion
Survey

https://ec.europa.eu/eusurvey/publication/RIPE88OpenSourceWGSurvey
Survey

- What makes a project trustworthy?
  - "Software you consider **mission critical** in your deployment"
- Secure deployment practices
- Risk mitigation practices
Survey

- Audience – bias
- Operators "who care"
- Presumably experts
  - RIPE Open source WG
  - RIPE DNS WG
  - dns-operations list @ DNS-OARC
  - Internet Systems Consortium's public channels
- 71 answers
How do you build confidence? #1

- documentation: 63%
- active and helpful mailing list: 56%
- releases are frequent/recent enough: 55%
- the software is already familiar to me: 48%
- versions are maintained for long enough: 44%
- we conduct thorough testing of the software: 35%
- there is more than one regular committer: 35%
- history of CVEs: 23%
How do you build confidence? #2

- Number of open, unresolved issues: 23%
- Popularity (e.g. stars on GitHub): 20%
- Financial sponsors of the project are identified: 17%
- Adequate packaging options: 13%
- Project test suite: 13%
- Software development process: 11%
- Published roadmap: 7%
- Badges on the project's homepage: 1%
How do you verify signatures?

- Package manager software does that automatically: 51%
- Make sure there is a PGP-verified path to the signature: 24%
- Check that .sig files match the tarball: 11%
- No: 10%
- Don't know: 4%
Do you inspect source code?

- no: 75%
- yes - manually: 20%
- don't know: 6%
- yes - with automated checks: 3%
How do you install software?

- install packages from OS: 49%
- install packages from the project: 24%
- compile from source: 18%
- install packages from support organization: 6%
- install packages from a third party: 3%
How do you mitigate upgrade risks?

- can rollback deployments easily: 68%
- only deploy stable versions: 59%
- test internally: 56%
- incremental deployment: 30%
- check for outstanding CVEs: 25%
- have support contracts: 18%
- wait to see other users' reactions: 15%
- wait couple of maintenance releases: 13%
- never deploy a .0 version: 10%
- other: 6%
How do you test before production?

- Test in production - it's been fine so far: 46%
- Test in lab with an artificial load: 42%
- Run some tests we developed: 39%
- Test in lab with a copy of real traffic: 34%
- Run the unit tests included with the source distribution: 10%
- Don't know: 3%
A DNS server – example project
# BIND 9 in numbers

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First commit</td>
<td>1998</td>
<td>26 years ago!</td>
</tr>
<tr>
<td>C code</td>
<td>263 000</td>
<td>lines (w/o tests or comments)</td>
</tr>
<tr>
<td>Automake</td>
<td>3 143</td>
<td>lines</td>
</tr>
<tr>
<td>Autoconf</td>
<td>1 839</td>
<td>lines</td>
</tr>
<tr>
<td>M4</td>
<td>1 626</td>
<td>lines</td>
</tr>
<tr>
<td># of authors</td>
<td>50+</td>
<td>in the current codebase</td>
</tr>
<tr>
<td># config knobs</td>
<td>325+</td>
<td>some are context dependent</td>
</tr>
<tr>
<td># CVEs</td>
<td>130</td>
<td>mostly DoS</td>
</tr>
</tbody>
</table>

* attribution of old code is hard – squash & merge model
Existing code – audit

- Who knows what's in there?!
  - 26 years!

- Security audit in 2023
  - 1 CVE, 2 medium severity, 6 low, 23 "nits" …
    - Low-level bugs
Audit limits

- No DNS-protocol level bugs found
  - By auditors – non-DNS experts
    - Meanwhile …

<table>
<thead>
<tr>
<th>CVE #</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2023-50868</td>
<td>Preparing an NSEC3 closest encloser proof can exhaust CPU resources</td>
</tr>
<tr>
<td>2023-50387</td>
<td>KeyTrap - Extreme CPU consumption in DNSSEC validator</td>
</tr>
<tr>
<td>2023-6516</td>
<td>Specific recursive query patterns may lead to an out-of-memory condition</td>
</tr>
<tr>
<td>2023-5680</td>
<td>Cleaning an ECS-enabled cache may cause excessive CPU load</td>
</tr>
<tr>
<td>2023-5679</td>
<td>Enabling both DNS64 and serve-stale may cause an assertion failure …</td>
</tr>
<tr>
<td>2023-5517</td>
<td>Querying RFC 1918 reverse zones may cause an assertion failure when …</td>
</tr>
<tr>
<td>2023-4408</td>
<td>Parsing large DNS messages may cause excessive CPU load</td>
</tr>
</tbody>
</table>
There is more than one regular committer
The software is already familiar to me
Versions are maintained for long enough
We conduct thorough testing of the software
There is more than one regular committer
History of CVEs
Self-imposed policies

- Coding & review procedures
- OpenSSF software quality badge
  - Lots of non-technical requirements
- ISC software defect and security vulnerability disclosure
- ISC CVSS scoring guidelines
- A lot of invisible work
BIND 9 vs. survey – processes

- Number of open, unresolved issues: 23%
- Popularity (e.g., stars on GitHub): 20%
- Financial sponsors of the project are identified: 17%
- Adequate packaging options: 13%
- Project test suite: 13%
- Software development process: 11%
- Published roadmap: 7%
- Badges on the project's homepage: 1%
BIND 9 new code

- Peer review in GitLab
  - Very few external contributions
- Automated tests
  - Continuous integration in GitLab
  - Extra things "on side"
Automated tests

- Unit tests
- Integration
- Fuzzers
- Interoperability
- Stress
- Performance …

GCC Code Coverage Report

<table>
<thead>
<tr>
<th>Metric</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines</td>
<td>77.1 %</td>
</tr>
<tr>
<td>Functions</td>
<td>85.5 %</td>
</tr>
<tr>
<td>Branches</td>
<td>55.6 %</td>
</tr>
</tbody>
</table>

incl. 12 204 assertions, 65.3 % without them
Continuous integration

For **main**

- **Scheduled**
  - 112 jobs
  - 69 minutes 53 seconds, queued for 14 seconds

**Pipeline**
- Needs: 112
- Jobs: 112
- Failed Jobs: 1
- Tests: 6321
BIND 9 vs. survey – tests

- number of open, unresolved issues: 23%
- popularity (e.g. stars on GitHub): 20%
- financial sponsors of the project are identified: 17%
- adequate packaging options: 13%
- project test suite: 13%
- software development process: 11%
- published roadmap: 7%
- badges on the project's homepage: 1%
Peer review
Peer review

include/dns
- message.h +3 -20
- win32
- libdns.def.in +0 -1
- message.c +56 -107
- rbtdbc.c +40 -129
- resolver.c +0 -1
- tkey.c +4 -18
- tsig.c +3 -6
- xfrin.c +0 -2
- zone.c +0 -3
- ns
- client.c +12 -38
- query.c +48 -21
- xfrout.c +0 -2

8✓ Approve

lib/dns/rbtdbc.c

+ static const unsigned char maptoupper[256] = {
};
Release ... err, try again ...

ISC (https://fosstodon.org/@iscdotorg)
@ISCdotORG

Ok, this is embarrassing. Please don't install the @bind9 updated versions we posted yesterday. Someone reported an error - we left out a LETTER of the ALPHABET in a streamlined routine. We will be removing the new versions and reposting after we have a chance to retest.

5:24 AM · Jun 18, 2021
**Automated tests – limits**

**W or w characters in domain names are altered to "\000"**

- Closed
- Issue created 2 years ago by Sean Zhang

**Summary**

We recently upgraded our bind9 from `1:9.16.16-2+ubuntu18.04.1+isc+1` to `1:9.16.17-1+ubuntu21.04.1+isc+1` and start experiencing some wildcard names not being resolved. The resolver will return `servfail`. After some troubleshooting we found that:

Under certain conditions (reproducible), the name in answer will not match the name in question. Found this issue reproducible with following conditions:
Peer review – limits

The matrix is square! It must be fine!
BIND 9 Release process

Release Checklist

Before the Code Freeze

1. (QA) Review 5 patches on top of current open-source versions: git checkout master; git patch; git push origin master
2. (QA) Inform Support and Marketing of impending release (and give estimated release date).
3. (QA) Ensure there are no permanent test failures on any platform. Check public and private scheduled pipelines.
4. (QA) Check changes from (AB/MET) to see if the scheduled pipelines to verify there is no unexpected performance drop for any protocol.
5. (QA) Check PullFails to ensure there has been no unexpected drop in performance for the versions being released.
6. (QA) Check whether all issues assigned to the release milestone are resolved.
7. (QA) Ensure that there are no outstanding merge requests in the private repository (subscription editor only).
8. (QA) Ensure all merge requests marked for backporting have been indeed backported.
9. (QA) Announce the freeze to the community that the code freeze is in effect.

Before the Tagging Deadline

10. (QA) Inspect the current output of the tag-version-check-tests to verify that no unexpected backport or incompatible change was introduced in the current release cycle.
11. (QA) Ensure release notes are correct, ask Support and Marketing to check them as well. Example
12. (QA) Add a release marker to EXAMPLES: Examples: 7.15, 5.16
13. (QA) Add a release marker to EXAMPLES: SE (Subscription Edition only). Example
14. (QA) Update BIND 9 version in [versiontag]-decoder v1.29 or version 1.30.
15. (QA) Convert configuration using Automator or bind-cfg-arg (1.16).
16. (QA) Update SRV settings for all maintained branches to disable mixing them: public, private
17. (QA) Tag the repositories in the private repository (git tag -a "BIND 9.x.y" v.x.y.z).

Before the ASN Deadline (for ASN Releases) or the Public Release Date (for Regular Releases)

18. (QA) Check that the formatting is correct for the HTML version of release notes.
19. (QA) Check that the formatting of the generated man pages is correct.
20. (QA) Verify SRV lab CI results for the tags created and sign-off on the releases to be published.
21. (QA) Update SRV lab settings for all maintained branches to allow merging them again: public, private
22. (QA) Prepare (using [versiontag]-decoder) and merge changes resulting the release notes and updating the version string for each maintained branch.
23. (QA) Reuse the Subscription Edition branches (including recent release prep commits) on top of the open source branches with updated version strings.
24. (QA) Announce the freeze to the community that the code freeze is over.
25. (QA) Request signatures for the tarballs, providing their location and checksums. Ask signers on Mattermost.
26. (Signers) Ensure that the contents of tarballs and tags are identical.
27. (Signers) Validate the checksums, sign tarballs, and upload signatures.
28. (QA) Verify tarball signatures and check tarball checksums again. Run publish-local.sh on repo to tag and pre-publish.
29. (QA) Prepare the patchset(s) under review for each security release (if applicable)
30. (QA) Pre-publish ASN and/or Subscription Edition tarballs so that packages can be built.
31. (QA) Build and test ASN and/or Subscription Edition packages (in cloudsmith branch in private repo, example)
32. (Marketing) Prepare and send out ASN emails (as outlined in the CVE checklist if applicable).

On the Day of Public Release

1. (QA) Wait for clearance from Security Officer to proceed with the public release (if applicable)
2. (QA) Place tarballs in public location on FTP site.
3. (QA) Inform Marketing of the release, providing FTP links for the published tarballs.
4. (QA) Use the Publiner project to prepare a release announcement email.
5. (Marketing) Publish links to downloads on ISC website. Example
6. (Marketing) Update the BIND 9 information document in SF with download links to the new versions. (If this is a security release, this will have already been done as part of the ABI process)
7. (Marketing) Update the Current Software Version document in the SF portal if any stable versions were released.
8. (Marketing) Send the release announcement email to the announce mailing list (and to dev-users if a major release – example).
9. (Marketing) Announce release on social media sites.
10. (Marketing) Update the wiki entry for BIND.
11. (Support) Ask the new releases to the security mailing list in the Knowledge Base.
12. (Support) Update fixes in case of with fixes support customers.
13. (QA) Build and test any outstanding private packages in private repo. Example
14. (QA) Rebuild public RPMs. Example: centos
15. (QA) Rebuild build简直就是ubuntu packages.
16. (Development) Update Changelog file and make sure push is synchronized to GitHub. Docker Hub should pull it up automatically. Example
17. (QA) Ensure all new tags are annotated and signed. git tag -a [new-release-vn].v.x.y.z
18. (QA) Push tags for the published releases to the public repository.
19. (QA) Using merge tag push, merge published-release tags back into their relevant development/maintenance branches.
20. (QA) Ensure these tags exist and only exist on the newest version of each release cycle.
21. (QA) Sanitize confidential issues which are assigned to the current release milestone and do not describe a security vulnerability, then make them public.
22. (QA) Leave confidential issues which are assigned to older release milestones and describe security vulnerabilities, then make them public if appropriate.
23. (QA) Update QA tools used in QA lab (e.g, Black, PreInt, SipClaim) by modifying the relevant SipClaim file.
24. (QA) Run a pipeline to rebuild all images used in QA lab.
25. (QA) Update release-x.x.x-320 with the upcoming release information.
BIND 9 release process

- Check list of changes that went in (again)
- Polish docs
- Run tests (again)
- Generate tarball
  - Check reproducibility
- Sign
- Publish
- Build packages
BIND 9 tarball checks

- Git ⇒ tarball reproducibility
  - [Link to GitHub repository and script](https://gitlab.isc.org/isc-projects/BIND9/-/blob/main/util/release-tarball-comparison.sh)
  - 100 lines
  - easy enough for independent review
BIND 9 tarball signing

- Dedicated VM
  - takes tarball from Gitlab
  - requests GPG signature
- Signer – person
  - SSH into the VM
  - forwards GPG agent socket
BIND 9 vs. survey – signatures

- Package manager software does that automatically: 51%
- Make sure there is a PGP-verified path to the signature: 24%
- Check that .sig files match the tarball: 11%
- No: 10%
- Don't know: 4%
**BIND 9 package build**

- Our RPM packages build in Gitlab
- Copr, Launchpad, Docker, etc. – manual
BIND 9 vs. survey – packages

1. **install packages from OS** - 49%
2. **install packages from the project** - 24%
3. **compile from source** - 18%
4. **install packages from support organization** - 6%
5. **install packages from a third-party** - 3%
<table>
<thead>
<tr>
<th>Feature</th>
<th>Team priority</th>
<th>Survey priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE frequency</td>
<td>#1</td>
<td>#8</td>
</tr>
<tr>
<td>CI &amp; automated tests</td>
<td>#1</td>
<td>#11</td>
</tr>
<tr>
<td>Code reviews &amp; standards</td>
<td>#1</td>
<td>#13</td>
</tr>
</tbody>
</table>
Discussion
Thank you!

• Main website: https://www.isc.org
• Software downloads: https://www.isc.org/download or https://downloads.isc.org
• Presentations: https://www.isc.org/presentations
• Main GitLab: https://gitlab.isc.org