Threat Model - Crates.io

This threat model covers crates.io, the main repository and package website for the Rust Programming Language. This model was crafted in collaboration with the crates.io and Infrastructure teams, as well as corporate and personal users.

Actors

User

Developer or organization using the rust for projects.

Author

An actor publishing a crate to crates-io

Trusted Author

An actor within a widely used crate.

Project Member

A member of a Rust Project team. For purposes of this document, this can refer to rustc, libs, rustup, crates-io, cargo or infra.

Assets

crates-io

https://crates.io
https://github.com/rust-lang/crates.io/

Index Repository

https://github.com/rust-lang/crates-io-index

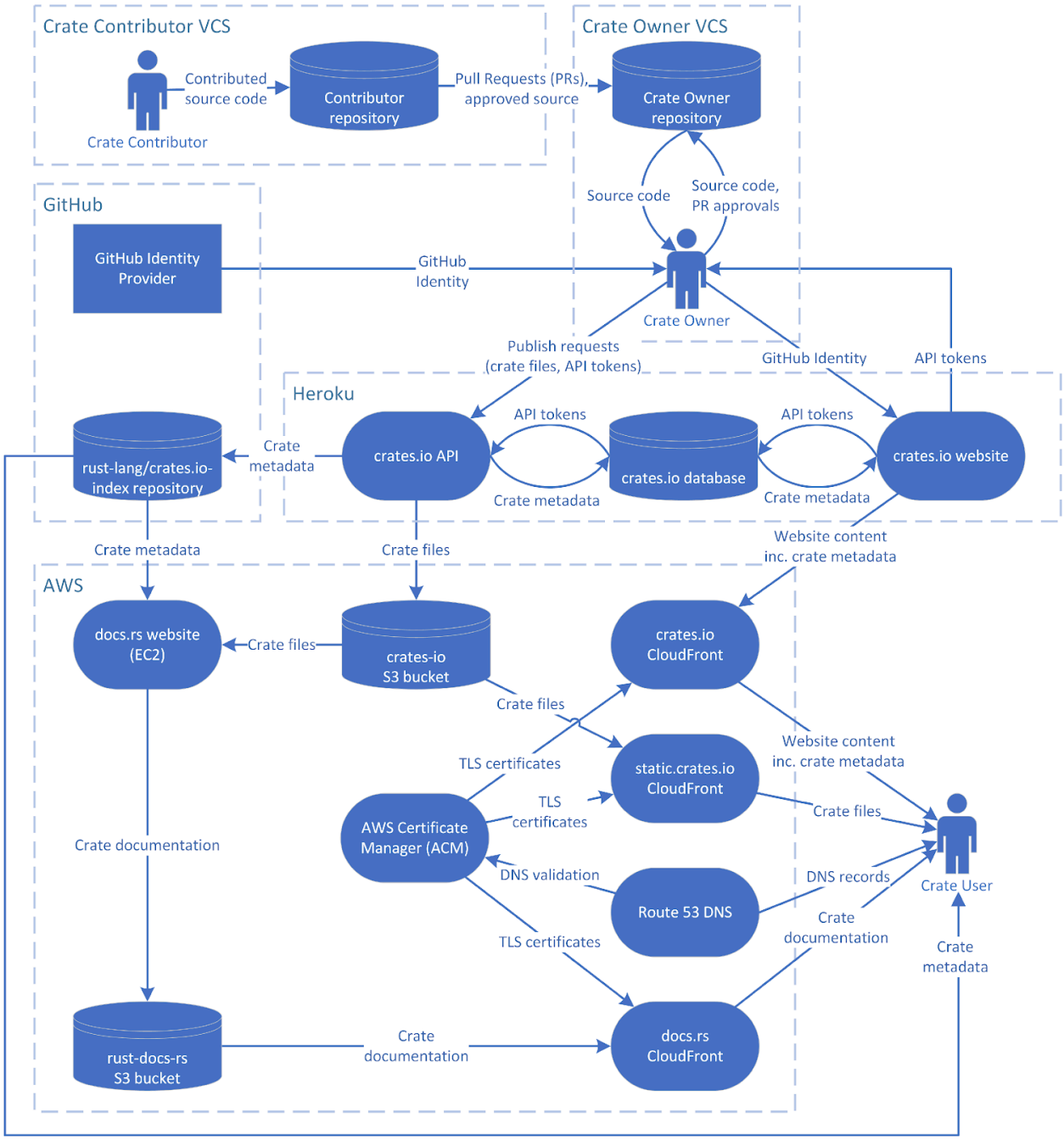
crate storage

Currently an S3 bucket and Fastly mirror, storing the raw tgz files of crates.

author secret

{HOME}/.cargo/credentials file local on any author's system which is validated on the server side of crates-io. In the case of private repositories, this takes on a different meaning and is out of the scope of this document.

Crate Publishing Model



In this model of publishing crates via crates.io, we start with Crate Contributors who do not have permission to publish new versions of a crate to crates.io. Crate Contributors first publish their contributions to crates as source code changes in their own repository (Contributor repository), then submit a Pull Request (PR) to the Crate Owner's repository. The Crate Owner reviews the contributed source code in the PR and, when satisfied, approves the PR to merge the change into the Crate Owner repository. GitHub is the predominant Version Control System (VCS) used by Crate Owners but repositories may be hosted in any VCS.

When the Crate Owner wants to publish a new crate, or a new version of an existing crate they own, they need a crates.io API token that they can use to authenticate to the crates.io API. The Crate Owner generates crates.io API tokens using the crates.io website, which uses GitHub as an OAuth identity provider for authentication. The Crate Owner downloads the crates.io API token and stores it locally for subsequent use. The crates.io website also stores the token in the crates.io database to authenticate future API requests bearing that token. Crates.io API tokens currently do not expire automatically and must be explicitly revoked via the crates.io website. Only permitted Crate Owner identities can upload new versions of existing crates or "yank" specific crate versions to remove them from the crates.io index. Even after a version is removed from the index, Crate Users that have already taken a dependency on that specific version can still download and build it, so "yanks" do not constitute complete removal. Crate permissions are managed via the crates.io website and stored in the crates.io database.

The Crate Owner uses the Cargo tool to publish to crates.io. The request made by Cargo includes the Crate Owner's crates.io API token for authentication, a tar archive of the crate source code files (*tarball*), and metadata such as the current local git commit hash and a repository URL specified by the Crate Owner. The contents of the tarball are not matched against the contents of the repository by crates.io.

After the crates.io API successfully authenticates the Crate Owner via the supplied API token, it checks that they are authorized to publish to that crate, i.e., no crate yet exists with that name, or they are a permitted Crate Owner publishing a new version of an existing crate. If authorized, the API commits an update to the crates.io index repository in GitHub with the crate metadata, stores the metadata in the crates.io database, and uploads the crate files to the crates-io S3 bucket.

The docs.rs website watches the crates.io index repository in GitHub for new commits. When a new commit is found, docs.rs fetches the new crate files from the crates-io S3 bucket and builds the documentation for the crate using a nightly build of the Rust compiler inside a Docker container to sandbox the build.

The Crate User uses the Cargo tool (not depicted) to download the crates.io index from GitHub and the crate files from the crates-io S3 bucket. The crates.io index includes SHA256 hashes of the crates files which are validated by Cargo. The crates.io website provides an interface where Crates Users can discover crates and view details of crates and owners (crate metadata). The crates.io website on Heroku, the docs.rs website on EC2, and the static crate files in the crates-io S3 bucket are each served to Crates Users by their own CloudFront distributions. TLS server certificates for Crates Users to authenticate the CloudFront endpoints are provided by ACM with Route 53 domain validation. The codebases for docs.rs, crates.io and Cargo are all written in Rust, and each has a dependency tree of third-party crates.

Threat Scenarios

1. Infrastructure compromise

Severity: High, likelihood: Low, Complexity: High

Statement

We currently do not have adequate SEIM coverage within the Rust Project infrastructure, and specifically the hosted portions of the crates.io infrastructure.

Impact

This lack of visibility means a compromise on the crates.io website, CI/CD paths or the CDNs hosting crates files can be undertaken without us actively catching such a compromise. This means an attacker with access to any of these systems will have the ability to lateral move, compromise packages or end users with little to no evidence of the activity from our logs.

Mitigations

- Infra patching improvements
- Infra monitoring improvements
- crates.io deployment improvements

2. Token theft

Severity: High, likelihood: Medium, Complexity: Low

Statement

Token theft not only allows for an attacker to compromise specific crates or developers within the ecosystem. This attack currently has a high susceptibility to `worming` due to the interconnected nature of contributors within our ecosystem.

Impact

Any crates which a compromised token has access to publish can be maliciously modified by an attacker. Additionally, given the interconnected nature of authors we suspect many high value targets exist which will allow such a theft to be wormed. That is to say - malicious code in any given high value crate could be injected to steal further tokens, and thus allowing for code injection into more crates with malicious code, etc.

Mitigations

- crates.io - Admin & Moderation Improvements
- crates.io - Token Scopes

3. Cross-site Scripting Attacks via README

Severity: High, likelihood: Low, Complexity: Medium

Statement

Crates.io currently renders user supplied input via README files being rendered to the end user during browsing of crates. This attack surface relies on the security and XSS preventions inherent in the MD rendering scheme being utilized within crates.io. No further effort is needed by the crates.io team at this time, however it is recommended that these libraries be monitored for vulnerabilities at a higher level of scrutiny.

Impact

We consider the likelihood of such an attack as low, given the minimal attack surface and increased scrutiny these entrypoints have.

4. CDN Manipulation

Severity: High, likelihood: Low, Complexity: High

Statement

We currently have minimal integrity checking occurring across the CDN infrastructure of the project and the deployment pipeline for crates. This means that a compromise of the CDN aspect of our infrastructure will lead to the undetectable modification of crates.

Impact

We consider the likelihood of such an attack as low, given that we currently centralize our CDN efforts within the Infra team. However, we consider this a high value target for future attacks as the ecosystem grows; and an active threat which must be considered while discussions of PKI, Crate Signing and others are being actively explored and developed.

Mitigations

- Infra Team reliability improvements
- crates.io mirroring
- Crate Signing

5. Index Manipulation

Severity: High, likelihood: Low, Complexity: High

Statement

We currently have minimal integrity checking occurring across the Index repository of the project and the deployment pipeline for crates, and how the index is interacted with via bors automation. This means that a compromise of bors or the Index aspect of our infrastructure will lead to the undetectable modification of crates or even redirection of download URLs.

Impact

We consider the likelihood of such an attack as low, given that we currently centralize our Index to Github. However, we consider this a high value target for future attacks as the ecosystem grows; and an active threat which must be considered.

Mitigations

- GitHub Security and Audit Logs

6. Man-in-the-Middle attack - Authentication and Publishing

Severity: High, likelihood: Low, Complexity: High

Statement

A common method of attack via more sophisticated and nation state actors is to man-in-the-middle the downloads of packages and sources and modifying them in transit. Although SSL is used for all communications to our ecosystem, no additional validation occurs at the crate or author level to guarantee the provenance and authenticity of the downloaded source. We believe efforts in PKI and Crate Signing (TUF et al) can help mitigate such an attack.

Impact

In the event a man-in-the-middle attack occurs against a crate, an attacker is capable of injecting or intercepting malicious code into a crate. Additionally, if an index update is intercepted then an attacker could manipulate the direct download URLs for any given crate.

Mitigations

- Crate Signing
- Rust PKI

7. Loss of a Project Member

Severity: Low, likelihood: Medium, Complexity: *N/A*

Statement

Crates.io was the first role in which the Foundation hired full time staff to join the team and provide maintenance of the project. This shift has dramatically reduced the risk of losing volunteers within the team and the impact this could have.

Impact

In the event of a loss of one of these authors, access may be lost and manual efforts will need to be undergone to allow the continuation of their work across any given crate or project. This is currently a cumbersome manual process.

Mitigations

- Documentation Improvement

8. Typosquatting

Severity: High, likelihood: High, Complexity: Low

Statement

We have already seen active attacks within the ecosystem involving TypoSquating attacks. This attack involves publishing malicious code with a crate similarly named to a popular and/or well trusted crate already existing, commonly with subtle variations to the name to deceive users. Especially in our ecosystem, this kind of attack may

commonly involve the inversion of `_` and `-`, or the changes of plural or addition of `rs`. Of course, a large amount of variation exists in these attacks.

Impact

The main purpose of such an attack path is to deceive a victim into downloading a malicious package to be executed. As such, in the event such an attack is realized we anticipate it will be part of an attack chain compromising one or more other attacks iterated here.

Mitigations

- The crates.io team has implemented the first stages of typosquatting detection, and efforts are under way in implementing automation for these solutions

8. Denial of Service

Severity: High, likelihood: High, Complexity: Low

Statement

The crates.io team is regularly responding to performance degredations caused by abnormal usage or edge cases involving crates.io. The majority of these issues stem from the inherent design and architecture of crates.io, its heavy reliance on direct database queries and a lack of caching across the site.

Impact

In the event of a successful or accidental denial of service attack, only some aspects of crates.io become unavailable at this time due to the segmented design. Specifically - the index currently exists on github and the sparse index, while crate files themselves can be directly downloaded from the CDNs in the event of a crates outage. However, publishing and other activities are disrupting when the service of crates.io degrades.

Mitigations

- We recommend efforts be focused on providing for the scalability of crates.io over the coming years

Ongoing Mitigations

Admin & Moderation Improvements

The Rust Foundation is currently funding software engineering efforts to develop a full admin and moderation framework for administrative use on crates.io. This effort is aimed to allow us to actively respond to attacks in a faster manner. Additionally, this effort is aimed to free up crates.io team resources to allow for further security work.

Recommendations

Crate Signing (TUF/Sigstore)

Other languages are in the early stages of implementing Sigstore, a methodology and framework for providing granular provenance across a package repository. Such a solution should also be implemented for crates.io.

Disaster Recovery Planning & Implementation

The Infra & crates.io team should plan and implement disaster recovery and resilience practices, in anticipation of many threats which could cause the crates.io and ecosystem to be down for a prolonged period of time. These efforts should be implemented in a way for us to immediately fault over to backup systems in the event of a compromise or denial of service attack, preserving the reputation of the Rust language and project.

Documentation & Reproducibility

Much of the knowledge involving the internal workings of Rust, its ecosystem and crates.io is considered “tribal” in nature. We should move forward with rigorous documentation of all associated portions of these systems to allow for easier onboarding of contributors and to mitigate the risk of the loss of power contributors.

Auditable & Centralized Logging

Many deficiencies currently exist within our logging; at both the system and application level. These raise serious risks in the event of an attack or compromise - hampering or completely removing our ability to properly respond to an attack. The infra team should design and deploy a centralized logging solution which can be properly leveraged for security monitoring and response across the ecosystem's infrastructure.