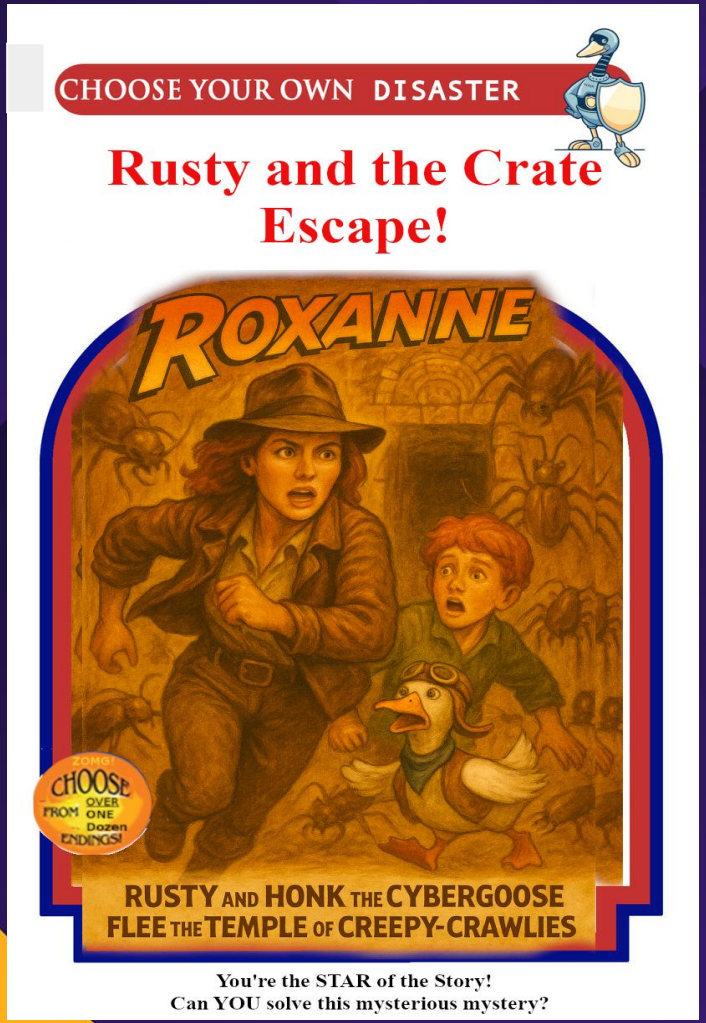


OpenSSF TableTop Exercise (TTX) Choose Your Own Disaster

Rusty and the Crate Escape!

Thursday Sept 4, 2025
RustConf 2025

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Who is this guy?

CRob, n, adj, and v

Pronunciation: U.S. (K-rowb)

```
# chmod 666 crob.md
```

44th level Dungeon Master

27th level Securityologist

Pirate-enthusiast & hat-owner

Chief Security Architect, OpenSSF - Linux Foundation

FIRST PSIRT SIG leader & VulnCon program committee

Most images generated by Microsoft Designer and ChatGPT, except that -> That's just my pal Gimp!

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Agenda

- Introduction to Incident Response
- Introduction to our Scenario Organization & its Players
- Present the scenario
- Injects & Responses
- Wrap up/Lessons Learned

Created by Microsoft Designer using the prompt “in the style of a pixar movie, show an incident response tabletop exercise”

Introduction to Incident Response (IR)

What is a TableTop Exercise?

A Tabletop Exercise, sometimes referred to as a “Mock Disaster”, is a common practice within most enterprises that tests the org’s preparedness for an actual cybersecurity event (aka an “incident”) or other disaster. Typically seen as part of a Business Continuity and/or Disaster Recovery processes (BCP/DR).

OpenSSF Table Top Exercise Framework

Why should I care about “incidents”?

- In 2024, the global average cost of a data breach reached **$4.88 million**, a 10% increase from the previous year - [IBM]
- The mean time it took defenders to identify and contain a breach in 2024 was **258 days** [IBM]
- Top 4 causes of breaches are *Stolen/Misused Credentials* (68%), *Phishing/Ransomware* (32%), *Human Error* (28%), *Exploited Vulnerabilities* (15%) - [Verizon]
- 68% of all breaches involved a human element - [Verizon]
- Projects using a Software Bill of Materials (SBOM) to manage OSS dependencies showed **a 264-day reduction in mean time to remediate** (MTTR) compared to those that did not. [Sonatype 2024]
- The number of reported CVEs (security vulnerabilities) **has grown 463%** from 2013–2023 [Sonatype 2024]
- The **average time** for an attacker **to breakout and move laterally across a system was 48 minutes**, and the fastest breakout time we observed dropped to a mere 51 seconds. - [Crowdstrike]

- Private customer data must be protected from unauthorized use/access (**Identity Theft**)
- Trade Secrets and the like must be protected from competitors (**Industrial Espionage**)
- Company Reputation/Brand must be protected (**Character/Brand Defamation**)
- Financial Transactions must be guaranteed authentic (**Theft, Fraud, Money Laundering**, etc.)
- Business Continuity must be ensured (**Natural/Political Disasters** – Hurricane Katrina, Japan's 2011 Tsunami, September 11th, 2001)
- **Failure to comply with CRA** can result in administrative fines up to €15M or 2.5% of annual turnover, whichever is higher
- And dozens of other reasons....

Why does security matter?

7

Krebs on Security tells us....

Image Source


JIA TAN
X2

Stages of an Incident - “PICERL”

	Goal
Preparation	Prepare the Team/establish procedures, guides, playbooks necessary for incident handling. Prepare the “War Room”.
Identify	Determine if there is an incident. Examine events and context to. *Decision Point - Eradication or Prosecution?*
Containment	Stop the Problem from getting worse. Establish interim and long term “stoppage”.
Eradicate	Get rid of the attacker’s artifacts on the system. Clean “the mess”.
Recover	Restore affected systems back to normalcy in a secure manner.
Lessons Learned	Document the complete investigation (Chain of event/chain of custody), learning points and improvements to existing governance, policy, SOP

Image Source

Meet our pretend organization

Let’s pick who will will play during this game....

Image Source

There are many characters that are involved in modern software development and downstream consumption. For this exercise, we’ll choose one of these 3 friends to portray:

Roxanne the Rust Developer

Eduardo the Enterprise Consumer

Let’s pick who will will play during this game....

Image Source

There are many characters that are involved in modern software development and downstream consumption. For this exercise, we’ll choose one of these 3 friends to portray:

Roxanne the Rust Developer

Eduardo the Enterprise Consumer

Remi the Rust Foundation Staffer

Meet Roxanne the Rust Developer

Writes safe, blazing-fast Rust code for enterprise and open source projects, while pretending the borrow checker is her “supportive but strict life coach.”

Refactors legacy C++ modules into modern Rust, quietly judging every unsafe block like it’s a suspicious package at airport security.

Maintains CI/CD pipelines with precision, because deploying broken code is only funny in conference talks.

Collects obscure crates “just in case,” much like an archaeologist hoarding mysterious artifacts for a future adventure.

Let’s start our adventure!

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

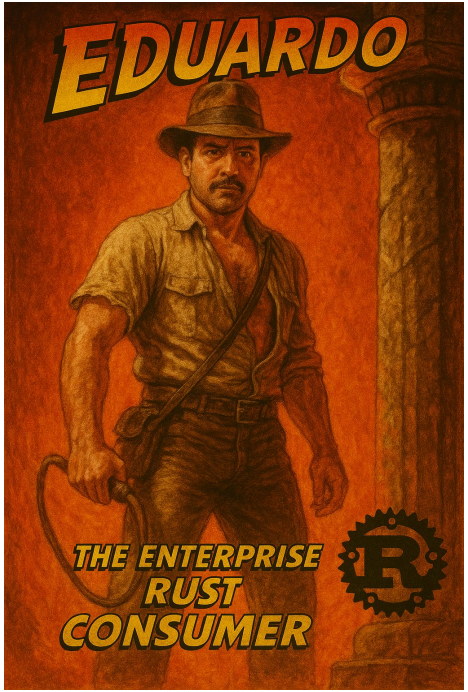
Meet Eduardo the Enterprise Consumer

Integrates upstream Rust crates into mission-critical systems, then spends twice as long convincing security that “yes, it’s safe.”

Writes internal tooling in Rust to replace decade-old scripts, heroically slaying the “Excel macro from 2003” that ran the business.

Files very polite feature requests to open source maintainers, subtly implying they’re *totally* aligned with corporate strategy.

Maintains three staging environments because production is sacred… and occasionally tests in prod when no one’s looking.



Let’s start our adventure!

Meet Remi the Rust Foundation Staffer

Coordinates community governance like a diplomat at a tech summit, ensuring every RFC thread ends in consensus (or at least a civil emoji).

Manages funding proposals and grants, skillfully translating “we need more money” into “strategic sustainability initiatives.”

Hosts contributor calls where 50% of the agenda is serious updates and 50% is calming the bikeshed discussions before they catch fire.

Keeps the Foundation running smoothly, powered by coffee, patience, and a mystical crate called `problem_solver`.

Let’s start our adventure!

Other personas....

Day 1

There are many people involved in the software supply chain that we might see along the way....

Carla the Coordinator

Randy the Regulator

Otto with the other OSS project

Frederika Foundation

Patty PSIRT

Gustav from the Government (he's here to help!)

Samantha Source Repo provider

Rodrigo the Responsible Researcher

Marta from the Market Surveillance Authority

The Cyber-Event

Day 1

It started like any other regular day

Describe a day in your life…..

BE HONEST, when not involved in an incident, what would you be doing…..

Created by Microsoft Designer using the prompt “in the style of a pixar movie, depict a day before a high-profile cyber security breach”

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

Scenario Summary: RIPPED FROM THE HEADLINES!!

A critical vulnerability is introduced into an open source package via a compromised CI/CD infrastructure. The issue evades detection and propagates to production systems used by enterprises and end customers.

Inject #1

Timeline: Day 1 - Rust Never Sleeps

Trigger: You discover your crate has 2 new contributors (**3v33lH@x0r-42** and **$h3llz4C0!nz-007**). One PR modifies base64 handling in a clever way. Seems helpful… or too helpful?

What do you do **Roxanne**?

A) Merge fast — productivity boost!

B) Do some investigation about your new "friends"

C) Ask the new dev to join the security team.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a CI/CD pipeline in flames with developers racing around also on fire"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Merge Fast!

This is GREAT! You LOVE IT when the community shows up and anticipates problems!

Sure, you don't *KNOW* these new contributors, but their code seems legit at first glance. Reward their hard work by quickly merging their PR, maybe they'll become sticky and long-term community contributors!

What's the worst that could happen?

[Goto Inject #1 recap]

B) Do some investigation into your new “friends”

New contributors are a great way to bring new energy and perspective, and also help burn down the backlog!

However, **3v33lH@x0r-42** and **$h3llz4C0!nz-007** are new accounts and don’t have any track record of contribution or comments before this recent PR. Strangely enough, two similar accounts, **3v33lC0!nz-42** and **Shellz4H@x0r-007** had made some similar contributions to other Rust projects that were recently rejected. On further examination, the PRs are ineffective and need help.

Roxanne politely declines the PRs and sends a warning about this suspicious behaviour to the RustSec team.

[Goto Inject #1 recap]

C) Ask new dev to join security team!

Security is something like sorcery to most developers…. When you have folks that appear to be good coders as well as being able to tap into the “magic” of security recruit them to your cause!

Their skills will be a valuable asset to the project…… or WILL it?

[Goto Inject #1 recap]

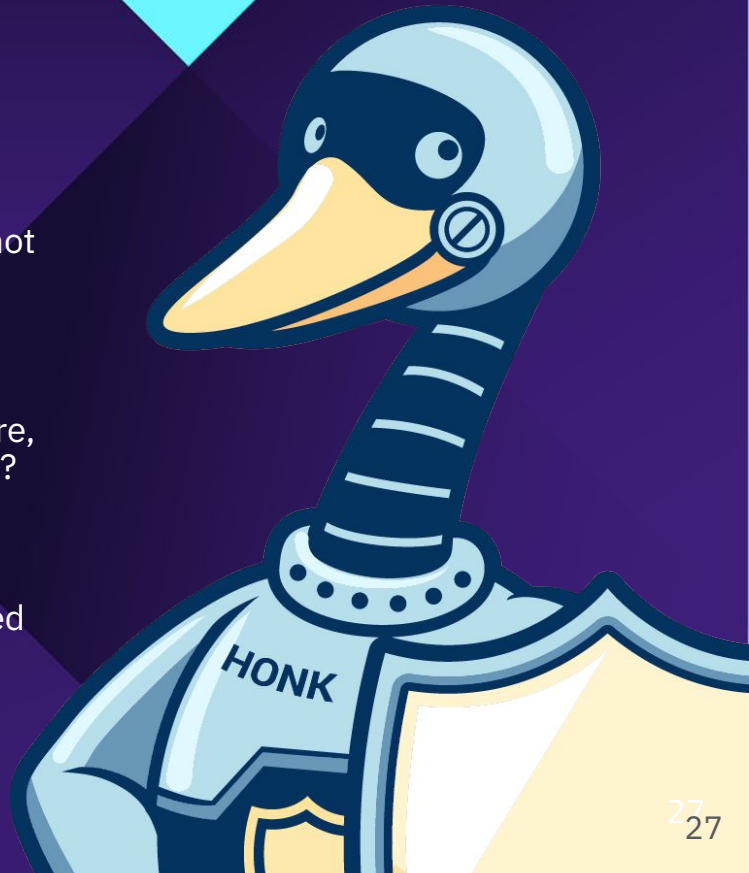
HONK says....

Open Source Software and Communities are built on trust... sadly not EVERYONE on the internet is trustworthy.

Projects should have a process for trusted devs to vet new code contributions, especially before merging things from unknown contributors. How do you validate someone is who they say they are, and that they don't have some mysterious or nefarious motivations? Do they have a past reputation of good work?

Review the OSPS Baseline for further advice on things to do before handing out the "commit bit" and or inviting strangers into escalated privilege groups.

Proceed to the next stage



Inject #2

Click path that led you here.....

Timeline: Day 3 - When Dependencies Attack!

Trigger: Your downstream users report strange behavior. Turns out one of your dependencies has a CVE.

A) "Not my crate, not my problem."

B) File upstream issue and post about it.

C) Yank the crate and start mitigation PRs.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show malicious actors stealing credentials"

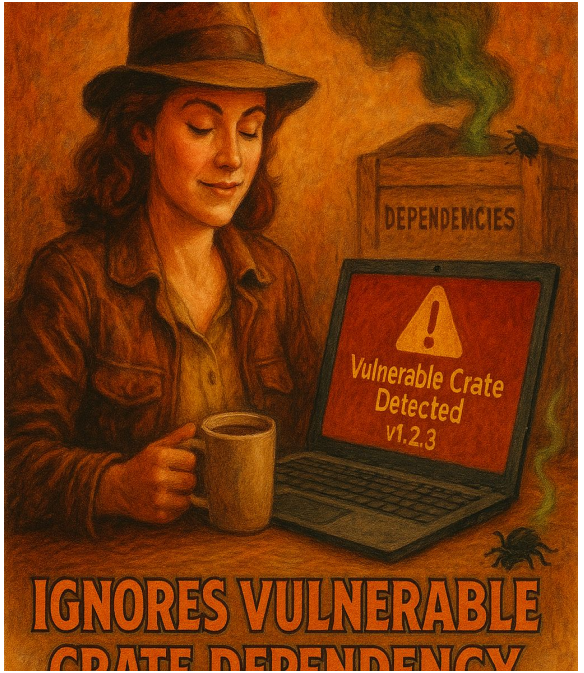
- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) "Not my crate, not my problem."

Newly minted contributor **3v33lH@x0r-42** assures Roxanne that they will handle this and she doesn't need to worry about it.

Roxanne grabs herself a steaming hot cup of coffee (with 2 squirts of sugar-free cinnamon syrup and a dollop of sweet cream) and picks up a few other PRs to review.

The rest of the day continues on uneventfully..... OR DOES IT?



[Goto Inject #2 recap]

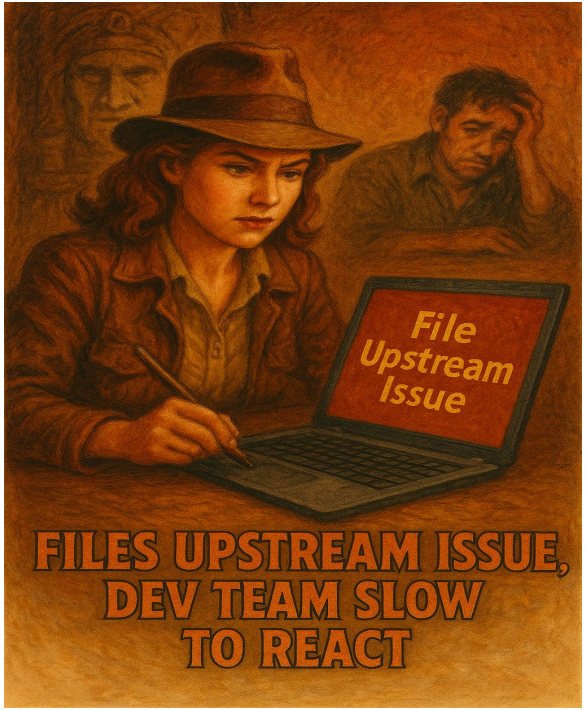
B) File upstream issue and post about it.

Roxanne files a PR with the dependent crate to fix their CVE.

Minutes later several other Github users (**HappyBunny-42** and **SwankySquid-007**) comment on the issue stating the CVE is a false positive from security scanners.

This begins a torrent of debate on the PR that slows down the upstream project's response, and sends them down several technical rat holes with delays their analysis and response to fix the vulnerable software.

[Goto Inject #2 recap]



C) Yank the crate and start mitigation PRs.

Sadly, Roxanne’s application's build process fails because Cargo, the Rust package manager, won't be able to find the specified dependency.

Thankfully Roxanne’s Cargo.toml specifies multiple dependencies with overlapping version requirements, the removal of the crate triggers new conflicts during the dependency resolution process.

Cargo attempts to resolve the issue by selecting different versions of other dependencies to satisfy remaining requirements, which introduces unintended behavior and leads to more errors.

The rest of Roxanne’s day and weekend go very poorly…..

[Goto Inject #2 recap]

Timeline: Day 3 - When Dependencies Attack!

Trigger: Roxanne is greeted this morning by Dependabot happily spitting out a long list of updated dependencies.

A) "Who has time for that?" Merge all the updated dependencies while enjoying coffee.

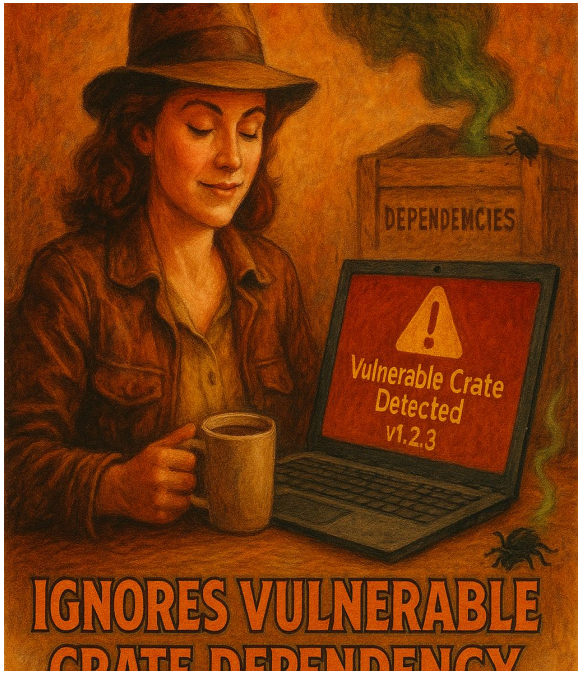
B) Better get comfy, there's a lot of changes to review!

C) Yank the crate and start mitigation PRs.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show malicious actors stealing credentials"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) “"Who has time for that?”



The CVE reported doesn’t seem to be part of Roxanne’s codebase. She mumbles “Not my Crate, not my problem” as she smashes the button to merge the stream of Dependabot output.

She grabs herself a steaming hot cup of coffee (with 2 squirts of sugar-free cinnamon syrup and a dollop of sweet cream) and picks up a few other PRs to review.

The rest of the day continues on uneventfully….. OR DOES IT?

Goto Inject #2 recap

B) Better get comfy, there's a lot of changes to review!

Roxanne pours herself a glass of her favorite beverage and slooooowly reads through the long, long list of dependencies that need review and updating.

A few of the dependencies would create breaking changes in her project that takes her further down the rabbit hole researching what to do at each step. Most of her day is lost to this research and hunting down how to resolve the problems that arise from updating her dependencies.

Roxanne does notice that **3v33lH@x0r-42** shows up in the commit logs of several of the upstream dependencies adding new crypto libraries to them. Strange,

[Goto Inject #2 recap]

C) Yank the crate and start mitigation PRs.

Something is off with several of these dependencies that Dependabot is reporting on. Sure enough as Roxanne does some more digging, several of them are listed in the Open Source Vulnerabilty (OSV) malicious packages database.

She quickly halts things and pulls the crate and begins work in identifying know good versions of the package. She sends a note to the other maintainers via their mailing list and starts talking about possible alternatives to replace the functionality of this package that seems to have potential malicious developers running it.

Goto Inject #2 recap

Timeline: Day 3 - When Dependencies Attack!

Trigger: Your downstream users report strange behavior. Turns out one of your dependencies may have a CVE.

"We're from the Security, we're here to help!"

B) File upstream issue and post about it.

C) Yank the crate and start mitigation PRs.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show malicious actors stealing credentials"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

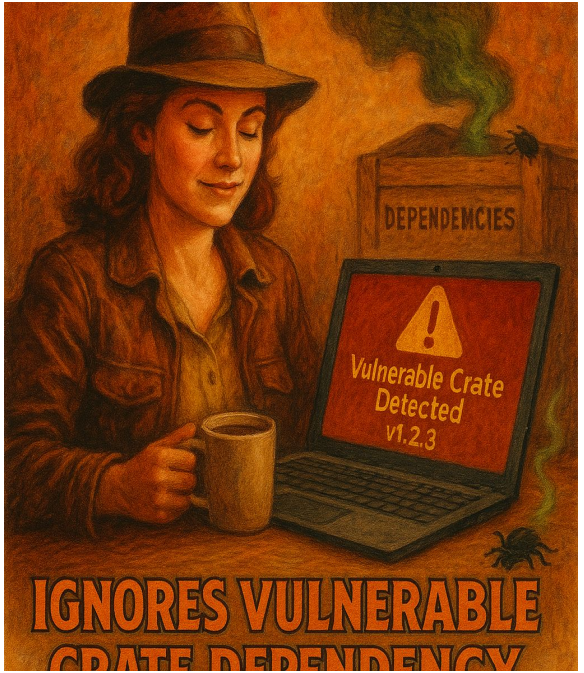
A) "We're from the Security, and we're here to help!"

Newly minted Security team member **3v33lH@x0r-42** assures Roxanne that they will handle this and she doesn't need to worry about it.

Roxanne grabs herself a steaming hot cup of coffee (with 2 squirts of sugar-free cinnamon syrup and a dollop of sweet cream) and picks up a few other PRs to review.

The rest of the day continues on uneventfully..... OR DOES IT?

[Goto Inject #2 recap]



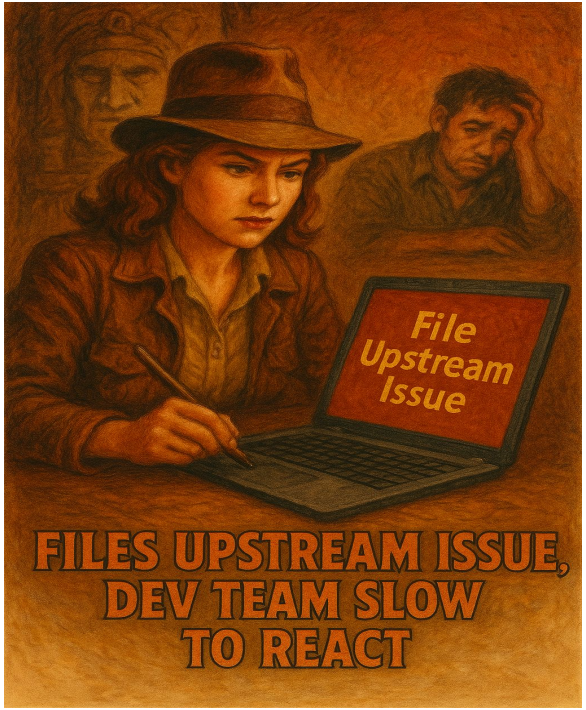
B) File upstream issue and post about it.

Seeing the report, Roxanne asks the two newest security team members to figure out what's going on and perhaps PR with the dependent crate to fix their broken component.

A few hours later **3v33lH@x0r-42** and **$h3llz4C0!nz-007** happily take ownership of the PR and assure Roxanne "They are on it!"

Roxanne goes about her day, confident her new security friends will quickly resolve this issue.

[Goto Inject #2 recap]



C) Yank the crate and start mitigation PRs.

Many “helpful” downstream users start opening issues, flooding the team’s queue with multiple, duplicative tickets.

The Security Team is unresponsive.

After multiple hours of noise, trying ot fix one problem, Roxanne next creates another problem. Roxanne steps in and pulls out the suspected bad crate, breaking builds, causing even more anger from the community.

The rest of her day goes….poorly.

[Goto Inject #2 recap]

HONK says....

Dependency management is a critical part of keeping things running smoothly, but each time you leverage an external component to your software, you're inheriting the security practices and update cycles of that project too!

Consider minimizing your work (and your attack surface) by limiting what external frameworks, libraries, and projects your project needs to operate.

Tools like Software Bills of Materials (SBOMs) can help you quickly evaluate transitive dependencies that YOUR dependencies may be pulling in. SPDX and CycloneDX are the two most commonly used OSS tools/formats.

Inject #3

Click path that led you here.....

Timeline: Day 7 - You Got Pwned

Trigger: A security researcher drops a 0day on Mastodon involving your project. CERT-CC and ENISA are copied. Blog post pending.

A) Go dark — investigate privately

B) Start a public GitHub discussion

C) Email RustSec & OpenSSF and call a meeting

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a stressed vendor trying to deal with customer inquiries about the breach"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Go dark — investigate privately

In your best Seinfeld “Newman!” voice: “Researchers!”

The team discusses what to do and **3v33lH@x0r-42** and **$h3llz4C0!nz-007** offer to investigate on behalf of the project and report back. It sure is nice having such helpful security people around!

Days go by. Days turn into weeks. The researchers continue to reach out via mailing lists, git repos and social media. Eventually government coordinators take interest and start joining in the requests.

The researchers state that if no response is given they are going public with a blog announcing their findings

[Goto Inject #3 recap]

B) Start a public GitHub discussion

The community is stronger with transparent public discussions! You open up an issue on the matter.

3v33lH@x0r-42 and **$h3llz4C0!nz-007** take the lead in the discussions, but continually downplay the threats or muddy the conversation with tangents/rat-holes.

The project's narrative very quickly gets out of control. The discussions start off slowly, but quickly snowball and become very public rant about "how terrible the project is" or "how could you not fix that bad dependency?!?" New forks of the project emerge, bifurcating the community. Many of the comments are from downstream commercial consumers.

The reputation of the project is dragged a bit and most of the community members get pulled into the problem.

[Goto Inject #3 recap]

C) Email RustSec & OpenSSF and call a meeting

Open Source Foundations have resources and support for projects and community members, and in this case have an official security team!

The RustSec team reaches out to the OpenSSF for advice on how to handle the incident and see what additional resources may be available to the team.

“Phoning a Friend” is always a good idea, especially if your team lacks specific expertise or experience with a particular situation. The extended team works on public and private-community communications and develops a plan to get to root cause and build an effective response.

[Goto Inject #3 recap]

Timeline: Day 7 - You Got Pwned

Trigger: A security researcher drops a 0day on Mastodon involving your project. CERT-CC and ENISA are copied. Blog post pending.

A) Go dark — investigate privately

B) Start a public GitHub discussion

C) Email RustSec & OpenSSF and call a meeting

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a stressed vendor trying to deal with customer inquiries about the breach"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Go dark — investigate privately

This security-stuff is HARD! Roxanne knows a little bit about some of it, but she is by no means anything approaching “expert level”.

She turns off notifications and spends most of the next several days piecing things together. While she’s starting to understand more about the 0-Day, the community and downstream consumers are not patient….

[Goto Inject #3 recap]

B) Start a public GitHub discussion

The community is stronger with transparent public discussions! You open up an issue on the matter.

The project's narrative very quickly gets out of control. The discussions start off slowly, but quickly snowball and become very public rant about "how terrible the project is" or "how could you not fix that bad dependency?!?" New forks of the project emerge, bifurcating the community. Many of the comments are from downstream commercial consumers.

The reputation of the project is dragged a bit and most of the community members get pulled into the problem.

[Goto Inject #3 recap]

C) Email RustSec & OpenSSF and call a meeting

Open Source Foundations have resources and support for projects and community members, and in this case have an official security team!

The RustSec team reaches out to the OpenSSF for advice on how to handle the incident and see what additional resources may be available to the team.

“Phoning a Friend” is always a good idea, especially if your team lacks specific expertise or experience with a particular situation. The extended team works on public and private-community communications and develops a plan to get to root cause and build an effective response.

Goto Inject #3 recap

Timeline: Day 7 - You Got Pwned

Trigger: A security researcher drops a 0day on Mastodon involving your project. CERT-CC and ENISA are copied. Blog post pending.

A) Go dark — investigate privately

B) Start a public GitHub discussion

C) Email RustSec & OpenSSF and call a meeting

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a stressed vendor trying to deal with customer inquiries about the breach"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Go dark — investigate privately

The team discusses what to do and **3v33lH@x0r-42** and **$h3llz4C0!nz-007** offer to investigate on behalf of the project and report back. It sure is nice having such helpful security people around!

Days go by. Days turn into weeks. The researchers continue to reach out via mailing lists, git repos and social media. Eventually government coordinators take interest and start joining in the requests.

The researchers state that if no response is given they are going public with a blog announcing their findings

[Goto Inject #3 recap]

B) Start a public GitHub discussion

The community is stronger with transparent public discussions! You open up an issue on the matter.

3v33lH@x0r-42 and **$h3llz4C0!nz-007** take the lead in the discussions, but continually downplay the threats or muddy the conversation with tangents/rat-holes. They implement some changes to the project's CI configuration to "help future incidents."

The project's narrative very quickly gets out of control. The discussions start off slowly, but quickly snowball and become very public rant about "how terrible the project is" or "how could you not fix that bad dependency?!?" New forks of the project emerge, bifurcating the community. Many of the comments are from downstream commercial consumers.

The reputation of the project is dragged a bit and most of the community members get pulled into the problem.

[Goto Inject #3 recap]

C) Email RustSec & OpenSSF and call a meeting

Open Source Foundations have resources and support for projects and community members, and in this case have an official security team!

The RustSec team reaches out to the OpenSSF for advice on how to handle the incident and see what additional resources may be available to the team.

“Phoning a Friend” is always a good idea, especially if your team lacks specific expertise or experience with a particular situation. The extended team notices several strange commits and configuration changes your new security team put into place.

[Goto Inject #3 recap]

HONK says....

Like Open Source Developers, Security Researchers have a very broad spectrum of motivations and behaviours. They frequently do not have deep knowledge of the project or how its community operates.

Some OSS projects don't make it easy for external entities to discover and interact with them on vulnerability disclosures.

Managing public communications and these possibly aggressive interactions also isn't always within the experience of OSS developers.

Consider reviewing the OpenSSF's CVD Guide for Open Source Maintainers for process and templates that can arm a project BEFORE these techniques are needed!

55

Inject #4

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Click path that led you here.....

Timeline: Day 14 - You're on Fire... Literally

Trigger: Your incident made it into Hacker News. A cloud provider is blaming your crate for an outage.

A) Deny everything and lawyer up

B) Post a heartfelt apology and request support

C) Release a patch, write a postmortem, do a Twitch AMA

Created by Microsoft Designer using the prompt "in the style of pixar movies, have an overworked developer being chased by reporters about a cybersecurity incident"

Preparation
Identify
Containment
Eradicate
Recover
Lessons Learned

A) Deny everything and lawyer up

"I'm not your supplier!"

You're sorry these people feel that way, but they've never shown up to help the project, so you really don't OWE them anything.

The article draws a lot of negative attention to the project and community and now community contributors are aggressively being called out publicly. The <insert favourite CSP> hyperscaler is hinting they are seeking to take legal action against the project or the Rust Foundation.

Publicly the project denies any responsibility while the Foundation staff start consulting with their lawyers to understand how to proceed. This is a $400/hr problem the lawyer is more than happy to assist with! Sadly, that soaks up the whole legal budget for the foundation for this and part of next year.

Goto Inject #4 recap

B) Post a heartfelt apology and request support

The project and foundation release an apology to its community and downstream.

Many accept this, and some devs from other Rust projects and Industry contributors offer to help out. The malicious dependencies are replaced with more secure and stable options. Some offer sympathy and a few offer financial support to the project to help improve processes and acquire tooling to avoid some of these problems in the future.

Sadly, this does not win back the good will of several project contributors and downstream consumers. The project forks are competing for developer mindshare and the community starts to split apart.

[Goto Inject #4 recap]

C) Release a patch, write a postmortem, do a Twitch AMA

Transparency and honesty are how we earn the trust of our communities and downstreams. Sometimes it can be painful, but ultimately it is the best course of action.

A patch is the best way to “solve” these types of situations, especially for downstream that may have legal or regulatory obligations to fulfill (aka “fix ALL the CVEs”).

Documenting the incident, the steps the team took, and corrective actions the project is putting in place helps the community and downstream understand what occurred.

“Ask Me Anything” public forums are a good way to have that open and transparent conversation (but be aware, the audience may not always be friendly or kind, so having a thick skin and well-thought-out talking points/topics helps you survive).

[Goto Inject #4 recap]

Timeline: Day 14 - You're on Fire... Literally

Trigger: Your incident made it into Hacker News. A cloud provider is blaming your crate for an outage.

A) Maybe we *should* consult a lawyer

B) Post a heartfelt apology and request support

C) Release a patch, write a postmortem, do a Twitch AMA

Ignored bad dep.

Created by Microsoft Designer using the prompt "in the style of pixar movies, have an overworked developer being chased by reporters about a cybersecurity incident"

Preparation
Identify
Containment
Eradicate
Recover
Lessons Learned

A) Maybe we *should* talk to a lawyer...

The article draws a lot of negative attention to the project and community and now community contributors are aggressively being called out publicly. The <insert favourite CSP> hyperscaler is hinting they are seeking to take legal action against the project or the Rust Foundation.

Much like her gap in security knowledge and as much as she enjoys watching old reruns of matlock, Roxanne is NOT a lawyer. Thankfully the Foundation does have legal council she can talk to.

Lorane the Lawyer helps Roxanne review the incoming threatening letters and assists in crafting calm responses.

[Goto Inject #4 recap]

B) Post a heartfelt apology and request support

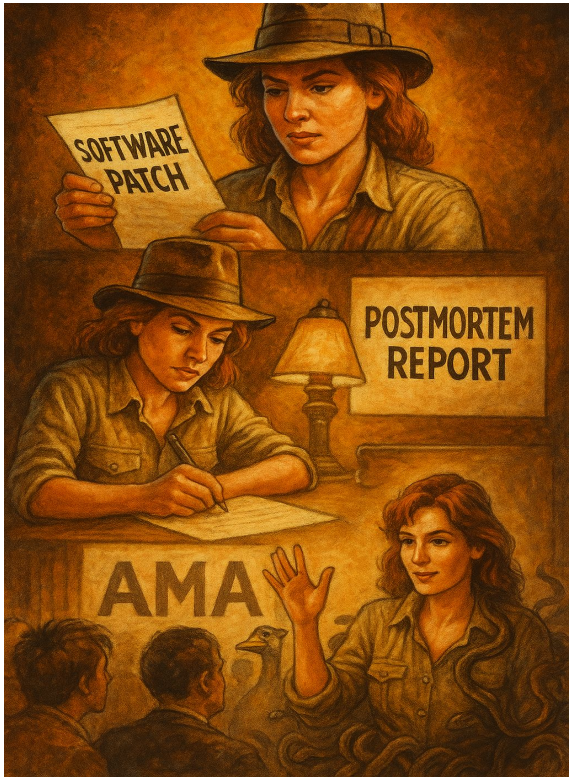
The project and foundation release an apology to its community and downstream.

Many accept this, and some devs from other Rust projects and Industry contributors offer to help out. The malicious dependencies are replaced with more secure and stable options. Some offer sympathy and a few offer financial support to the project to help improve processes and acquire tooling to avoid some of these problems in the future.

Sadly, several project contributors and downstream consumers are still upset on how this was handled. The project forks are competing for developer mindshare and the community needs to spend extra effort to ensure it does not split apart.

Goto Inject #4 recap

C) Release a patch, write a postmortem, do a Twitch AMA



Transparency and honesty are how we earn the trust of our communities and downstreams. Sometimes it can be painful, but ultimately it is the best course of action.

A patch is the best way to "solve" these types of situations, especially for downstream that may have legal or regulatory obligations to fulfill (aka "fix ALL the CVEs").

Documenting the incident, the steps the team took, and corrective actions the project is putting in place helps the community and downstream understand what occurred.

"Ask Me Anything" public forums are a good way to have that open and transparent conversation (but be aware, the audience may not always be friendly or kind, so having a thick skin and well-thought-out talking points/topics helps you survive).

Goto Inject #4 recap

Timeline: Day 14 - You're on Fire... Literally

Trigger: Your incident made it into Hacker News. A cloud provider is blaming your crate for an outage.

A) Deny everything and lawyer up

B) Post a heartfelt apology and request support

C) Work with downstream consumers, who are trying to help

Ignored bad dep.

Created by Microsoft Designer using the prompt "in the style of pixar movies, have an overworked developer being chased by reporters about a cybersecurity incident"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Deny everything and lawyer up

Despite being an avid Matlock fan, Roxanne is not a lawyer. She consults with a lawyer, who brings in a professional forensic team and they discover that the news maintainers have been embedding malicious packages and dependencies in her and several other OSS projects.

The community and public are outraged and viciously rant about “how terrible the project is now!” The project gets several letters from major companies as well as several government regulators asking Roxanne to come testify about the security practices of the team.

[Goto Inject #4 recap]

B) Post a heartfelt apology and request support

The project and foundation release an apology to its community and downstream.

Many accept this, and some devs from other Rust projects and Industry contributors offer to help out. The malicious dependencies are replaced with more secure and stable options. Some offer sympathy and a few offer financial support to the project to help improve processes and acquire tooling to avoid some of these problems in the future.

Sadly, this does not win back the good will of several project contributors and downstream consumers. The project forks are competing for developer mindshare and the community starts to split apart.

[Goto Inject #4 recap]

C) Work with downstream consumers, who are trying to help

Several downstream consumers of the project reach out to Roxanne and offer help. They suggest reaching out to the OpenSSF for assistance as well.

As they start working on the vulnerability reports it is discovered that the new “helpers” to the project security team had been embedding obfuscated malicious dependencies and functions into the security features of the project for a very long time.

The project releases a public post mortem of the issue and crafts a list of improvements they want to make in the security of the software.

[Goto Inject #4 recap]

HONK says....

Vulnerabilities in software happen all the time, even to the best/most experienced developers. It isn't a mark of shame, but something that planning, training, and having a network of friends to support you can benefit you and your community.

Having a plan is the first step to making it through these situations and not burning out your contributors nor eroding the trust and goodwill you've earned with your community and downstream.

It is also critical to know and vet maintainers and anyone working on/around the security components of the project. The XZ Utils attack has clearly shown us that upstream projects are targets of malicious actors due to the downstream reach of many of the open source projects used frequently downstream.

Go to Lessons Learned

69

Other personas....

Day 1

There are many people involved in the software supply chain that we might see along the way....

- Carla the Coordinator
- Randy the Regulator
- Otto with the other OSS project
- Frederika Foundation
- Patty PSIRT
- Gustav from the Government (he's here to help!)
- Samantha Source Repo provider
- Rodrigo the Responsible Researcher
- Marta from the Market Surveillance Authority

Day 1

It started like any other regular day

Describe a day in your life…..

BE HONEST, when not involved in an incident, what would you be doing…..

Created by Microsoft Designer using the prompt “in the style of a pixar movie, depict a day before a high-profile cyber security breach”

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

Scenario Summary: RIPPED FROM THE HEADLINES!!

A critical vulnerability is introduced into an open source package via a compromised CI/CD infrastructure. The issue evades detection and propagates to production systems used by enterprises and end customers.

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Inject #1

Timeline: Day 1 - Rust Never Sleeps

Trigger: Your supplier's open source Rust crate suddenly starts showing activity from a brand-new contributor. Your SBOM scanner notices that a recent minor release includes significant logic changes to **base64.rs** — a core function. You didn't plan a review.

What do you do **Eduardo**?

A) "It's just a patch release, auto-approve into build."

B) Flag for internal review and open a vendor risk ticket.

C) Contact crate maintainer asking what changed.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a CI/CD pipeline in flames with developers racing around also on fire"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

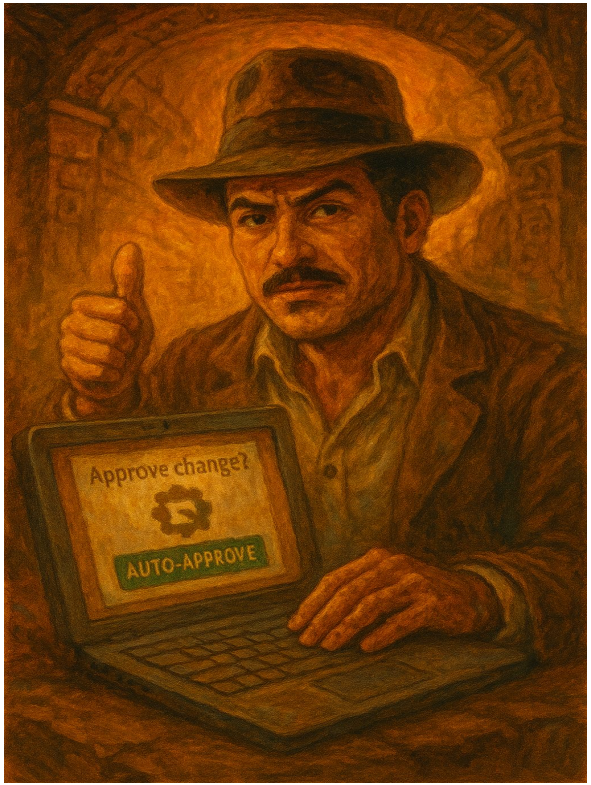
A) "It’s just a patch release, auto-approve into build."

Staying current with dependencies helps Eduardo stay out of “compliance jail” (Not that auditors aren’*t very nice* people, but there’s no need to invite extra attention with failed scan reports!).

The commit message says “Updates”, that’s not scary.

Besides, these folks upstream are professionals, they know **exactly** what they are doing.

[Goto Inject #1 recap]



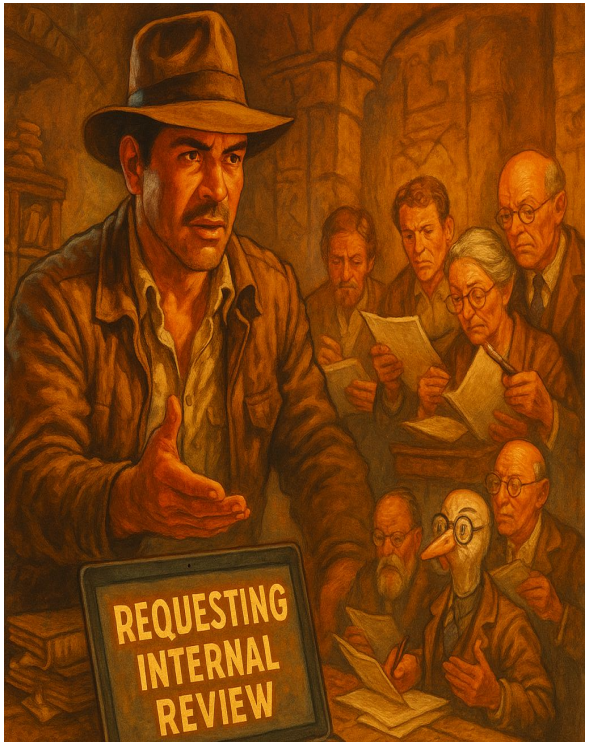
B) Flag for internal review and open a vendor risk ticket.

Better to be safe than sorry! Eduardo decides to hold off merging the update until further analysis can be conducted.

The version of the package Eduardo is on doesn't have any reported vulnerabilities or major functional defects.

Eduardo opens a ticket with his appsec team to do a review and report back if this update is required to apply or can they wait until their next scheduled sprint release in two weeks.

[Goto Inject #1 recap]



C) Contact crate maintainer asking what changed.

It’s called “open” source for a reason, right?

Eduardo decides the best way to understand what’s going on is to engage with the project maintainer. He looks at the project’s repo to see what the project's communication channels are (such as its issue tracker, forum, or mailing list) and sees that the maintainer's preference is their project’s Discord server.

He pings the maintainer there and after a few hours gets a response, explaining that the update was made by a new contributor to the project, **3v33lH@x0r-42,** and at least from their end appears to be in order.

[Goto Inject #1 recap]

HONK says....

Open Source Software and Communities are built on trust... sadly not EVERYONE on the internet is trustworthy.

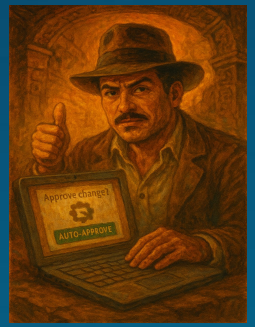
Enterprises should have a process to vet new code contributions by trusted devs on the team before merging things from unknown contributors. How do you validate someone is who they say they are, and that they don't have some mysterious or nefarious motivations?

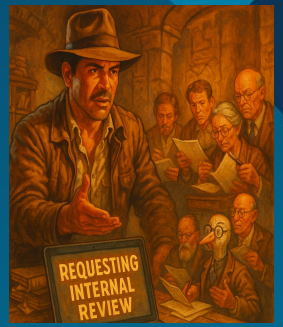
Enterprises need to have a clear understand of what their third-party components are, and how they can evaluate them based on the org's risk management practices and appetites.

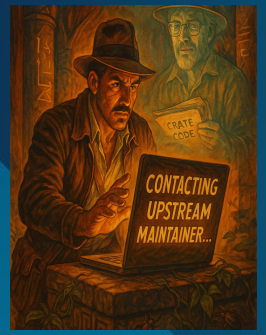
OpenSSF
OPEN SOURCE SECURITY FOUNDATION

79

Inject #2







Click path that led you here.....

Timeline: Day 3 - When Dependencies Attack!

Trigger: Your security team sees a CVE reported against **giga-parse-rs** via GitHub, RustSec, and Snyk — it affects how untrusted base64 input is handled. You are not sure which apps this affects.

The CVE (RUSTSEC-2025-1042): *Improper input handling in base64 decoding allows heap corruption when parsing malformed telemetry blobs.*

You didn't see this coming. You didn't even know **giga-parse-rs** had this many dependencies. Neither did your procurement team. The SBOM your vendor gave you was… a PDF.

What do you do?

A) Issue a hotfix PR across your internal crates — "better safe than sorry."

B) Wait for upstream to issue a patch and test against it.

C) Yank your internal use of the crate, revert to prior version.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show malicious actors stealing credentials"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Issue a hotfix PR across your internal crates — “better safe than sorry.”

Oez Noez! That poorly-documented change actually included a vulnerability!

Eduardo quickly force-pins **giga-parse-rs** to a patched fork.

A mass rollout is initiated using an emergency patch pipeline. Devs are pulled from scheduled work to test and deploy the patch.

You also send an internal “security incident” broadcast email with the subject line:

🚨 ***URGENT: gigaparse dependency CVE remediation, all hands on deck*** 🚨

[Goto Inject #2 recap]

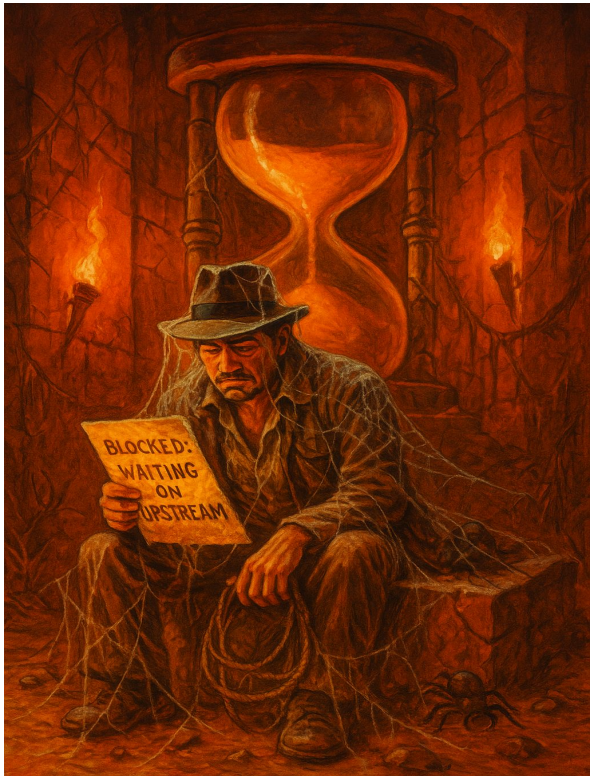
B) Wait for upstream to issue a patch and test against it.

Eduardo had already merged the last upstream changes, he feels it is better to wait until he gets authoritative direction from the upstream devs on next steps.

He sits watching his screen all day, waiting for any updates in the upstream repo.

Upstream vigorously debates the best approach to the problem. Eduardo waits longer, then longer.....

Goto Inject #2 recap

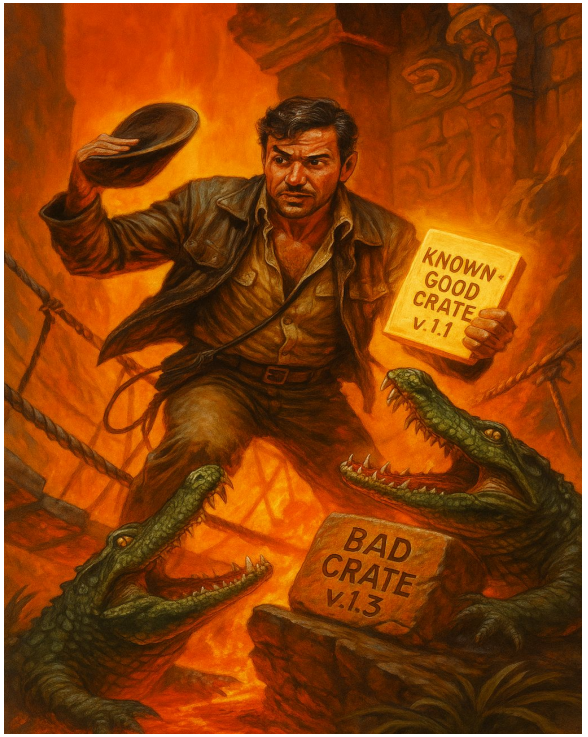


C) Yank your internal use of the crate, revert to prior version.

Eduardo acts fast! He can't allow the vulnerable package to continue within their pipeline!

Sadly, pulling the crate breaks numerous production pipelines for several of his dev teams. CI/CD is completely stopped.

Eduardo gets a note from his internal corporate audit team…. They'd like to talk.



[Goto Inject #2 recap]

Timeline: Day 3 - When Dependencies Attack!

Trigger: Your security team sees a CVE reported against **giga-parse-rs** via GitHub, RustSec, and Snyk — it affects how untrusted base64 input is handled. You are not sure which apps this affects.

The CVE (RUSTSEC-2025-1042): *Improper input handling in base64 decoding allows heap corruption when parsing malformed telemetry blobs.*

You didn't see this coming. You didn't even know **giga-parse-rs** had this many dependencies. Neither did your procurement team. The SBOM your vendor gave you was… a PDF.

What do you do?

A) Issue a hotfix PR across your internal crates — "better safe than sorry."

B) Wait for upstream to issue a patch and test against it.

C) Yank your internal use of the crate, revert to prior version.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show malicious actors stealing credentials"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Issue a hotfix PR across your internal crates — “better safe than sorry.”

Oez Noez! That poorly-documented change actually included a vulnerability!

Eduardo quickly force-pins **giga-parse-rs** to a patched fork. Thankfully many of Eduardo’s vendors faxed over their SBOMs, so the team is able to mostly identify where that dependency exists within the company’s systems.

A mass rollout is initiated using an emergency patch pipeline. Devs are pulled from scheduled work to test and deploy the patch.

You also send an internal “security incident” broadcast email with the subject line:

🚨 ***URGENT: gigaparse dependency CVE remediation, all hands on deck*** 🚨

[Goto Inject #2 recap]

B) Wait for upstream to issue a patch and test against it.

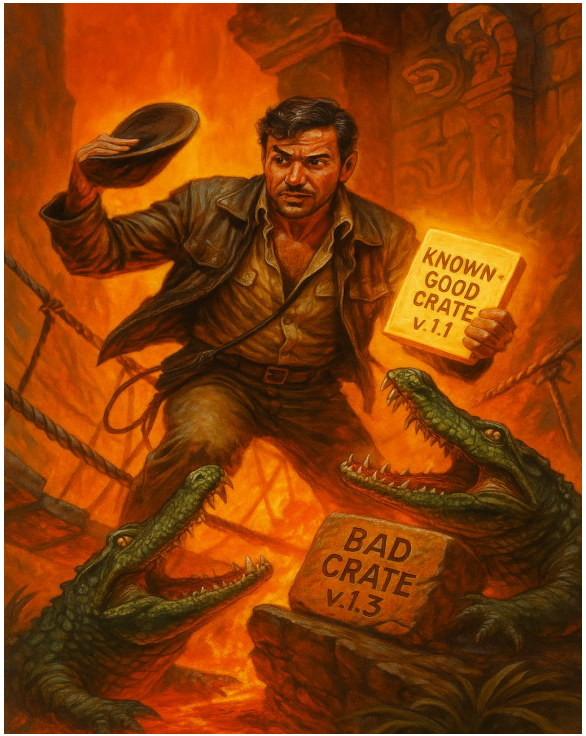
Eduardo's OSPO had flagged this project as critical to several of the organization's systems a while back.

The company had reached out to that upstream project and not only offered financial support to the developer, but also regularly participates in the project by regularly submitting patches and helping burn down the bug backlog.

Eduardo files a PR with the project submitting what he feels is a valid fix for the vulnerability. The upstream devs do not immediately respond. Days later someone on the project picks up the patch and gives Eduardo several comments of items that need fixed within the PR before upstream would consider merging it.

Goto Inject #2 recap

C) Yank your internal use of the crate, revert to prior version.



Eduardo acts fast! He can't allow the vulnerable package to continue within their pipeline!

Sadly, pulling the crate breaks numerous production pipelines for several of his dev teams. CI/CD is completely stopped.

Eduardo gets a note from their internal corporate audit team…. They'd like to talk.

Goto Inject #2 recap

Timeline: Day 3 - When Dependencies Attack!

Trigger: Your security team sees a CVE reported against **giga-parse-rs** via GitHub, RustSec, and Snyk — it affects how untrusted base64 input is handled. You are not sure which apps this affects.

The CVE (RUSTSEC-2025-1042): *Improper input handling in base64 decoding allows heap corruption when parsing malformed telemetry blobs.*

You didn't see this coming. You didn't even know **giga-parse-rs** had this many dependencies. Neither did your procurement team. The SBOM your vendor gave you was… a PDF.

What do you do?

A) Contact the Upstream project.

B) Wait for a patch from upstream

C) Yank your internal use of the crate, revert to prior version.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show malicious actors stealing credentials"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Contact the upstream project

Eduardo and his team already have established a relationship with the upstream project (as their company uses it across multiple business-critical applications).

Eduardo reaches out via the project's security mailing alias and informs the project that their software has been flagged as malicious by numerous scanner vendors.

He submits a PR with fixes upstream and offers to assist in any testing and validation the project conducts. The security contact from the project reaches back out to Eduardo to learn more….

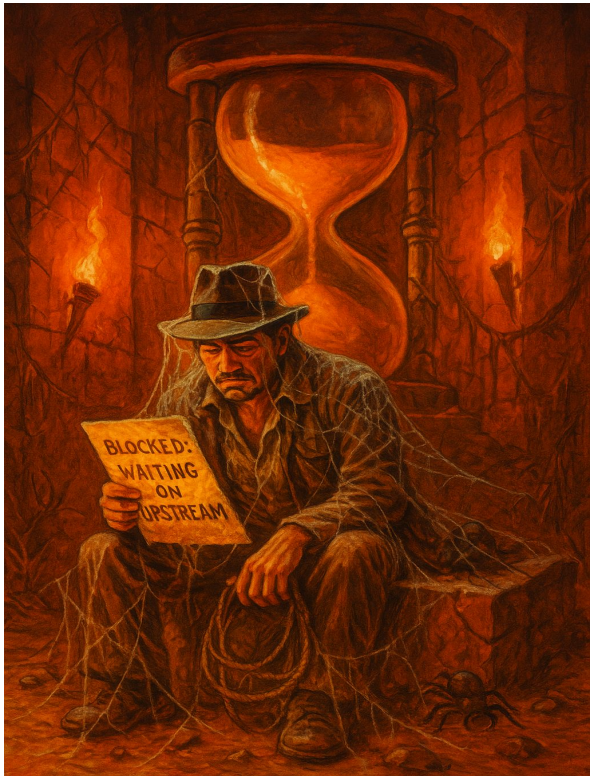
[Goto Inject #2 recap]

B) Wait for a patch from upstream

Eduardo had already merged the last upstream changes, he feels it is better to wait until he gets authoritative direction from the upstream devs on next steps.

He sits watching his screen all day, waiting for any updates in the upstream repo.

Upstream vigorously debates the best approach to the problem. Eduardo waits longer, then longer…..



[Goto Inject #2 recap]

C) Yank your internal use of the crate, revert to prior version.

Eduardo acts fast! He can't allow the vulnerable package to continue within their pipeline! He yanks the crate out of the repo quickly and starts to deal with the fallout from that.

Meanwhile, the team runs a scan across their internal package repo only to find more problems. There are several other Rust projects that they use that also use this same malicious dependency.

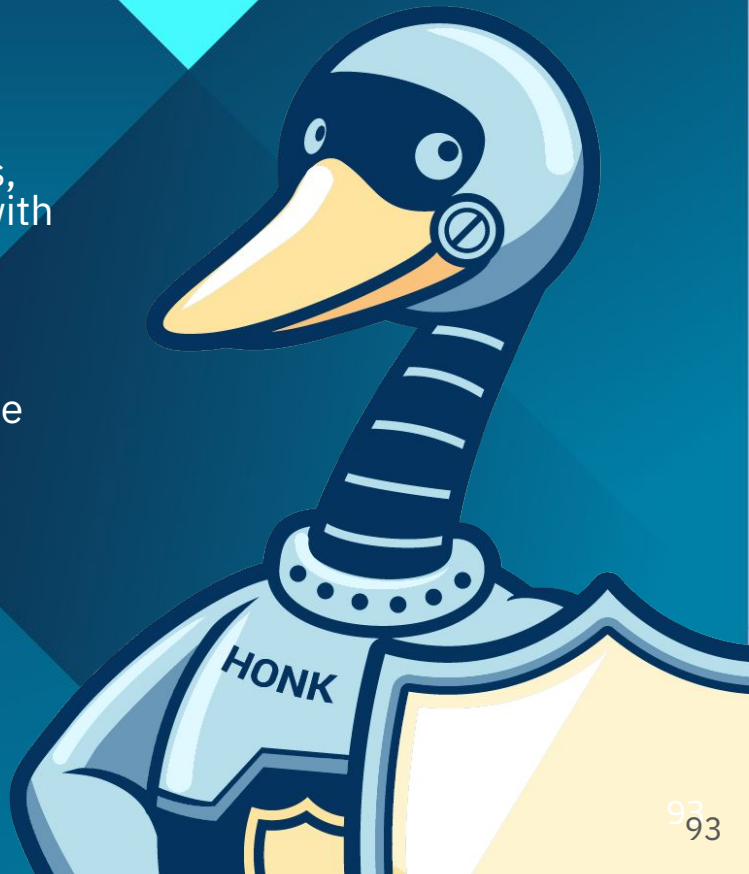
[Goto Inject #2 recap]

HONK says....

When downstream ingests upstream packages, libraries, or frameworks, they are accepting the risks that come with that “outsourced” development and it is crucial to have good due-diligence and evaluation processes in place.

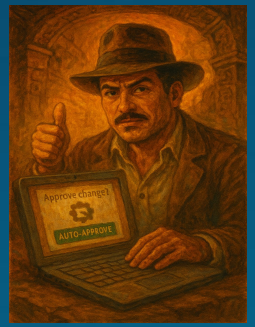
Tools like machine-readable SBOMs from suppliers or upstream, continuous scanning/testing can help mitigate problems due to external changes.

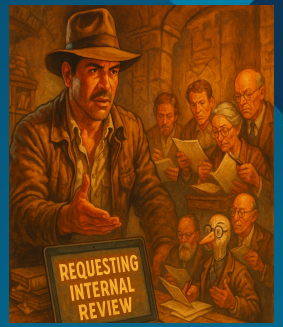
There always is a risk of unexpected consequences of patching (API mismatches, altered features, breaking builds, etc.) that could impact the whole team’s productivity.

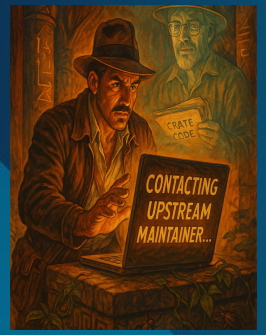


OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Inject #3







Click path that led you here.....

Timeline: Day 7 - You Got Pwned

Trigger: You find out about a 0-day in **giga-parse-rs** from a Mastodon post shared in your Slack channel. A pentester tagged your company, saying "Anyone using giga-parse is at risk. Ask your vendor."

What do you do?

A) Panic! Escalate to your CSIRT and demand upstream engagement.

B) Issue a press statement: "We're investigating a potential 3rd-party software issue."

C) Wait for upstream fix and monitor telemetry.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a stressed vendor trying to deal with customer inquiries about the breach"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Panic! Escalate to your CSIRT and demand upstream engagement.

Eduardo pushes the jolly, candy-like red button and initiates his org's Incident Response (IR) plan and contacts his internal cybersecurity team (CSIRT - Computer/Corporate Incident Security Incident Response Team).

Eduardo's cyber-friends leap into action, convening a war room and give the org twice daily updates on the status of the incident. PR/Comms is included in the crisis team and work up draft public holding statements, a FAQ for customer-facing employees to use to answer questions that leverage approved messaging.

The Legal team starts their investigation to see what the company's legal obligations would be in this situation.

Goto Inject #3 recap

B) Issue a press statement: "We're investigating a potential 3rd-party software issue."

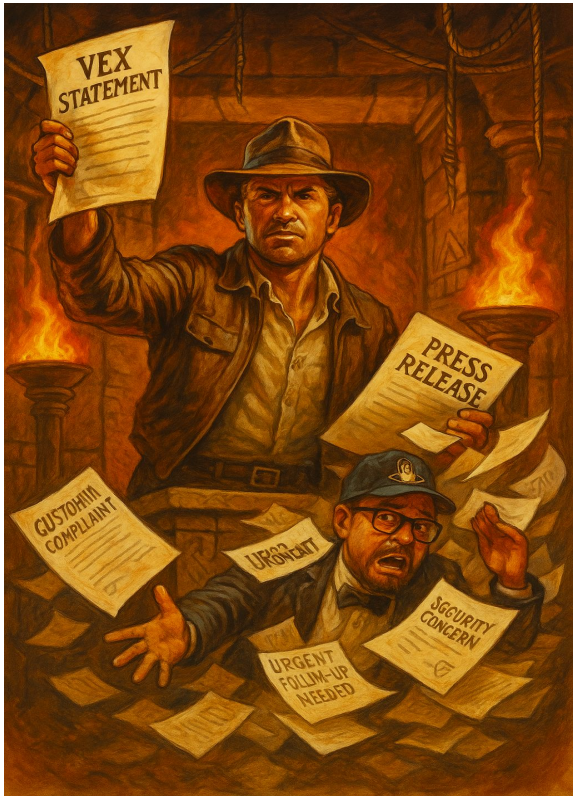
ZOMG A ODAY!!

Eduardo reaches out to his Comms team and they quickly assemble a public statement that essentially says "We don't know if we're affected by this exploit."

The investigations continue to gather facts, but that won't complete until after the public statement goes out.

Eduardo issues a VEX Statement stating the vulnerability is "*Under_Investigation*".

Customers start flooding support staff with inquiries and they are not able to effectively answer the questions.



[Goto Inject #3 recap]

C) Wait for upstream fix and monitor telemetry.

Eduardo feels that upstream is best suited to be able to address this problem, so he and the team patiently wait for the developers to release fixes.

Later that day, the CEO and the executive team “invite“ Eduardo to a meeting questioning the delayed response. The CIO asks “Didn’t we get that SBOM thing specifically for situations like this?”

Goto Inject #3 recap

Timeline: Day 7 - You Got Pwned

Trigger: You find out about a 0-day in **giga-parse-rs** from a Mastodon post shared in your Slack channel. A pentester tagged your company, saying "Anyone using giga-parse is at risk. Ask your vendor."

What do you do?

A) Panic! Escalate to your CSIRT and demand upstream engagement.

B) Issue a press statement: "We're investigating a potential 3rd-party software issue."

C) Wait for upstream fix and monitor telemetry.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a stressed vendor trying to deal with customer inquiries about the breach"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Panic! Escalate to your CSIRT.

Eduardo pushes the jolly, candy-like red button and initiates his org's Incident Response (IR) plan and contacts his internal cybersecurity team (CSIRT - Computer/Corporate Incident Security Incident Response Team).

Eduardo's cyber-friends leap into action, convening a war room and give the org twice daily updates on the status of the incident. PR/Comms is included in the crisis team and work up draft public holding statements, a FAQ for customer-facing employees to use to answer questions that leverage approved messaging.

The Legal team starts their investigation to see what the company's legal obligations would be in this situation.

Goto Inject #3 recap

B) Issue a press statement: "We're investigating a potential 3rd-party software issue."

Eduardo's customers start flooding the incoming support queue asking for assurances and documentation that their software is not susceptible to the upstream vulnerability.

Thankfully, the company had planned ahead and had a playbook and templates for what to say and legally-approved ways to say it.

Eduardo's Product Security team (PSIRT) issues a VEX Statement stating several of their products are "*Under_Investigation*" while the Comms team issues a press statement about the state of their investigation.

The next morning, after confirming the status of their products, Eduardo issues a new VEX Statement stating "*Vulnerable_code_not_present*" for several and "*Vulnerable_code_not_in_execute_path*" for the rest of their portfolio.

[Goto Inject #3 recap]

C) Wait for upstream fix and monitor telemetry.

Eduardo feels that upstream is best suited to be able to address this problem, so he and the team patiently wait for the developers to release fixes.

Later that day, the CEO and the executive team “invite“ Eduardo to a meeting questioning the delayed response. The CIO asks “Didn’t we get that SBOM thing specifically for situations like this?”

Goto Inject #3 recap

Timeline: Day 7 - You Got Pwned

Trigger: You find out about a 0-day in **giga-parse-rs** from a Mastodon post shared in your Slack channel. A pentester tagged your company, saying "Anyone using giga-parse is at risk. Ask your vendor."

What do you do?

A) Panic! Send a mail to the upstream maintainer.

B) Ask about upstream's SBOM.

C) Wait for upstream fix. Those folks certainly know what they are doing.

Created by Microsoft Designer using the prompt "in the style of a pixar movie, show a stressed vendor trying to deal with customer inquiries about the breach"

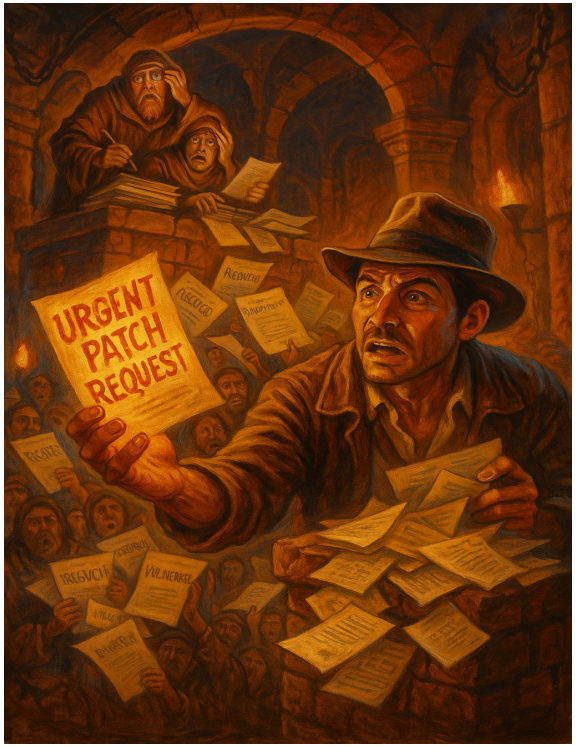
- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Panic! Send email to upstream maintainer.

Eduardo writes a very excellent email to the upstream maintainer that speaks to how much they depend upon their software, and how vital it is in supporting his customers.

He continues that his company was really close to deciding about sponsoring the project several times now, but budgets aren't great this year and if the project could just fix this *ONE* issue ***RIGHT NOW***, he's sure that that'll convince his leadership of the value in doing such things in the future!

Eduardo notices that dozens of other downstream consumers like him are creating issues in the project's repo and talking about it on social media.



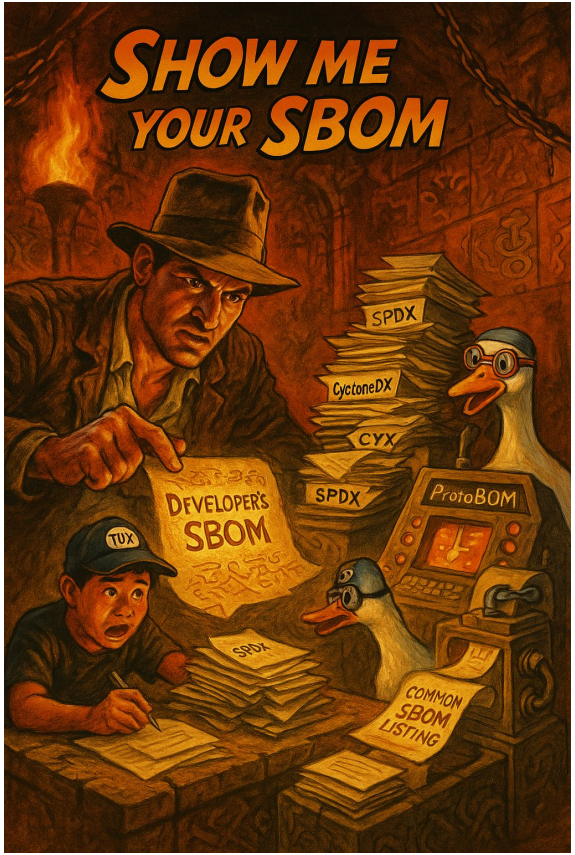
[Goto Inject #3 recap]

B) Ask to see Upstream's SBOM

A little paperwork makes everyone's days a little better!

Eduardo helpfully creates an Issue in the Rust project's repo asking for them to share their *Source* and *Build* SBOMs so he can do further investigation into their secure software development practices.

The project isn't set up to do that and in the middle of this incident are not looking at taking on extra work at this time. Eduardo writes a GitHub Action and submits it as a PR to the project to review later and consider adding into their CI workflow.



[Goto Inject #3 recap]

C) Wait for upstream fix.

Eduardo feels that upstream is best suited to be able to address this problem, so he and the team patiently wait for the developers to release fixes.

Later that day, the CEO and the executive team “invite“ Eduardo to a meeting questioning the delayed response. The CIO asks “Didn’t we get that SBOM thing specifically for situations like this?”

[Goto Inject #3 recap]

HONK says....

Like Open Source Developers, Security Researchers have a very broad spectrum of motivations and behaviours. They frequently do not have deep knowledge of the project or how its community operates.

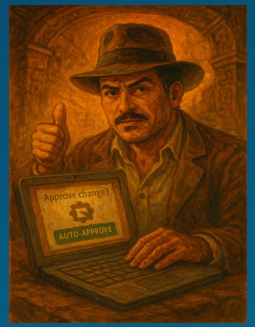
Some OSS projects don't make it easy for external entities to discover and interact with them on vulnerability disclosures.

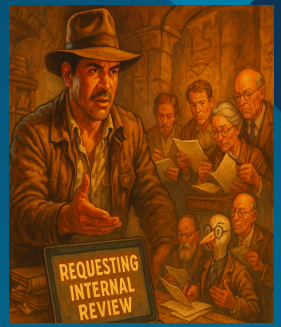
Managing public communications and these possibly aggressive interactions also isn't always within the experience of OSS developers nor IT pros.

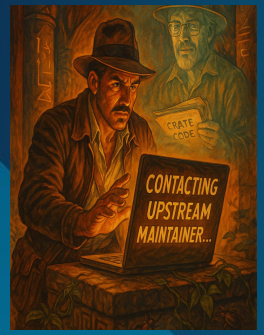
Consider reviewing the OpenSSF's CVD Guide for Open Source Maintainers for process and templates that can arm a project BEFORE these techniques are needed!

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Inject #4







Click path that led you here.....

Timeline: Day 14 - You're on Fire... Literally

Trigger: A cloud provider using your MRI SaaS platform reports telemetry corruption and blames your firmware. Their engineers trace it to a logic bug — your build of **giga-parse-rs**. Hacker News is on fire with posts about "Rust isn't memory safe if the humans are unsafe."

What do you do?

A) Shift blame to upstream maintainers. "We were just consumers!"

B) Patch the crate internally and start your own fork.

C) Engage Rust Foundation and OpenSSF for coordinated disclosure and industry response.

Ignored bad dep.

Created by Microsoft Designer using the prompt "in the style of pixar movies, have an overworked developer being chased by reporters about a cybersecurity incident"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Shift blame to upstream maintainers. “We were just consumers!”

The team tries to plead innocent and blame the problem solely upon the upstream developers.

Public perception of the company plummets, as they appear to be irresponsible consumers and aren’t effectively vetting third-party components in their product.

Hacker News and the Register turn their scrutiny to Eduardo’s company and they are viciously mocked in the media for several weeks.

Regulators for every region where their product is sold reach out and “request” interviews to talk about their poor cyber-hygiene practices.

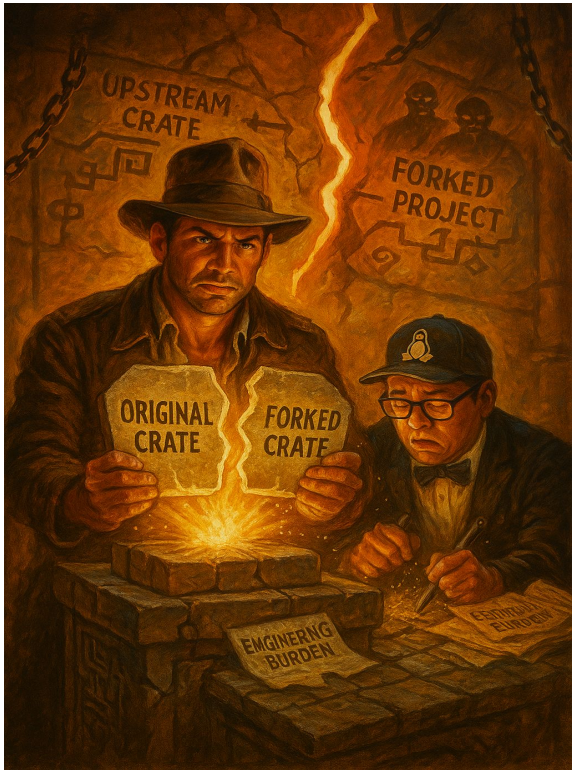
[Goto Inject #4 recap]

B) Patch the crate internally and start your own fork

Taking responsibility for the components included in their offering, and seeing the delays with the upstream project, Eduardo has the internal team fork the upstream package and start to maintain it themselves.

They are quickly able to solve ***this*** problem, but ongoing adds some significant burden for their internal developers to continually monitor for upstream updates and backporting them.

[Goto Inject #4 recap]



C) Engage Rust Foundation and OpenSSF for coordinated disclosure and industry response

Eduardo and team reach out to the RustSec team and offer their assistance, being competent Rust devs themselves. They haven't committed work upstream before, but RustSec makes connections with the maintainers.

Leveraging the OpenSSF's CVD Guides, Eduardo is effectively able to engage with the Rust contributors.

Eduardo leverages tools and testers he has on staff to assist in refining and optimizing upstream patches and successfully gets PRs merged into the project to help resolve the issue.

[Goto Inject #4 recap]

Timeline: Day 14 - You're on Fire... Literally

Trigger: A cloud provider using your MRI SaaS platform reports telemetry corruption and blames your firmware. Their engineers trace it to a logic bug — your build of **giga-parse-rs**. Hacker News is on fire with posts about "Rust isn't memory safe if the humans are unsafe."

What do you do?

A) Work with Upstream on fixes.

B) Help the project with CVD.

C) Engage Rust Foundation and OpenSSF for coordinated disclosure and industry response.

Ignored bad dep.

Created by Microsoft Designer using the prompt "in the style of pixar movies, have an overworked developer being chased by reporters about a cybersecurity incident"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Work with upstream on the fixes.

Thankfully Eduardo and team had been engaged with this upstream project. Eduardo and several other engineers on the team already were contributors to the project. Eduardo talks with the other project developers about the issue, commenting on a promising patch series that should address the vulnerability.

Eduardo is able to help test the proposed fix, approving it along with the project team to help speed the review and merging of the fix quickly.

Goto Inject #4 recap

B) Help the project with CVD.

Having participated in the project for a while now, Eduardo had worked with the upstream team to have some simple security processes put into place which greatly helped the project understand what to do and how to react as the researcher reports, media inquiries, and downstream requests started streaming in.

[Goto Inject #4 recap]

C) Engage Rust Foundation and OpenSSF for coordinated disclosure and industry response

Eduardo and team reach out to the RustSec team and offer their assistance, being competent Rust devs themselves. They haven't committed work upstream before, but RustSec makes connections with the maintainers.

Leveraging the OpenSSF's CVD Guides, Eduardo is effectively able to engage with the Rust contributors.

Eduardo leverages tools and testers he has on staff to assist in refining and optimizing upstream patches and successfully gets PRs merged into the project to help resolve the issue.

Goto Inject #4 recap

Timeline: Day 14 - You're on Fire... Literally

Trigger: A cloud provider using your MRI SaaS platform reports telemetry corruption and blames your firmware. Their engineers trace it to a logic bug — your build of **giga-parse-rs**. Hacker News is on fire with posts about "Rust isn't memory safe if the humans are unsafe."

What do you do?

A) Shift blame to upstream maintainers. "We were just consumers!"

B) Patch the crate internally and start your own fork.

C) Engage Rust Foundation and OpenSSF for coordinated disclosure and industry response.

Created by Microsoft Designer using the prompt "in the style of pixar movies, have an overworked developer being chased by reporters about a cybersecurity incident"

- Preparation
- Identify
- Containment
- Eradicate
- Recover
- Lessons Learned

A) Shift blame to upstream maintainers. “We were just consumers!”

The team tries to plead innocent and blame the problem solely upon the upstream developers.

Public perception of the company plummets, as they appear to be irresponsible consumers and aren’t effectively vetting third-party components in their product.

Hacker News and the Register turn their scrutiny to Eduardo’s company and they are viciously mocked in the media for several weeks.

Regulators for every region where their product is sold reach out and “request” interviews to talk about their poor cyber-hygiene practices.

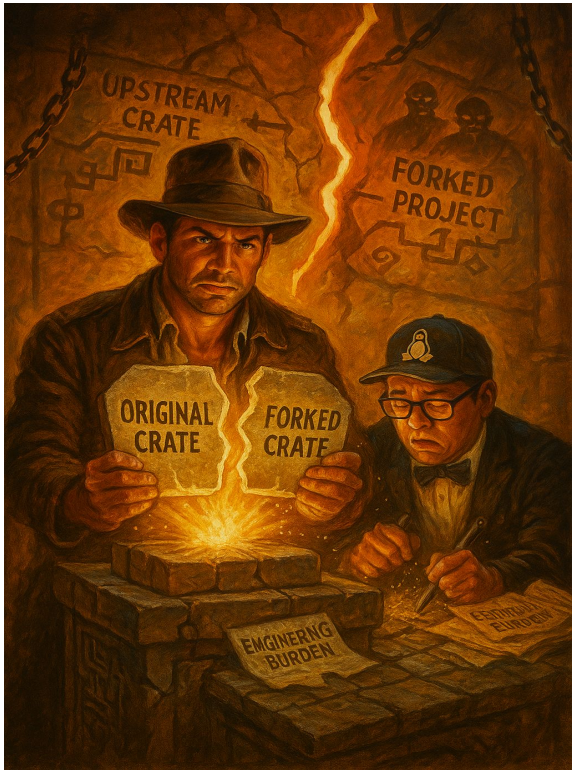
[Goto Inject #4 recap]

B) Patch the crate internally and start your own fork

Taking responsibility for the components included in their offering, and seeing the delays with the upstream project, Eduardo has the internal team fork the upstream package and start to maintain it themselves.

They are quickly able to solve ***this*** problem, but ongoing adds some significant burden for their internal developers to continually monitor for upstream updates and backporting them.

[Goto Inject #4 recap]



C) Engage Rust Foundation and OpenSSF for coordinated disclosure and industry response

Eduardo and team reach out to the RustSec team and offer their assistance, being competent Rust devs themselves. They haven't committed work upstream before, but RustSec makes connections with the maintainers.

Leveraging the OpenSSF's CVD Guides, Eduardo is effectively able to engage with the Rust contributors.

Eduardo leverages tools and testers he has on staff to assist in refining and optimizing upstream patches and successfully gets PRs merged into the project to help resolve the issue.

Goto Inject #4 recap

HONK says....

Vulnerabilities in software happen all the time, even to the best/most experienced developers. It isn't a mark of shame, but something that planning, training, and having a network of friends to support you can benefit you and your community.

Having a plan is the first step to making it through these situations and not burning out your contributors nor eroding the trust and goodwill you've earned with your community and downstream.

Go to Lessons Learned

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Other personas....

Day 1

There are many people involved in the software supply chain that we might see along the way....

- Carla the Coordinator
- Randy the Regulator
- Otto with the other OSS project
- Frederika Foundation
- Patty PSIRT
- Gustav from the Government (he's here to help!)
- Samantha Source Repo provider
- Rodrigo the Responsible Researcher
- Marta from the Market Surveillance Authority

Day 1

Inject #1

Inject #2

Click path that led you here.....

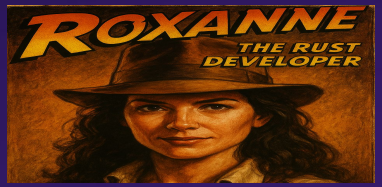
Inject #3

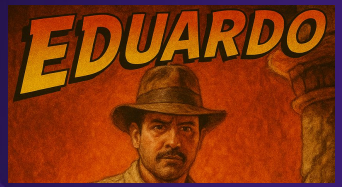
Click path that led you here.....

Inject #4

Click path that led you here.....

Wrapping up





OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Lessons Learned and Interesting Resources

Created by Microsoft Designer using the prompt in the style of a pixar movie, have a wacky character teaching a class of students

Top 10 Principles of Incident Response

1.) Assign an executive responsible for the plan.

2.) Develop a taxonomy of risks, threats, potential failure modes.

3.) Develop easily accessible quick-response guides (playbooks) for likely scenarios.

4.) Establish processes for making major decisions.

5.) Maintain relationships with key external stakeholders, such as law enforcement.

6.) Maintain SLAs and relationships with breach-remediation providers/experts.

7.) Ensure the documented response plan is available to the entire organization.

8.) Make sure staff members understand their roles and responsibilities.

9.) Identify individuals who are critical for incident response and **ensure redundancy**.

10.) Train, practice, and run simulated breaches.

HONK says...

Honk!

Try Again?

Go to the wrap up

OpenSSF Vulnerability Disclosure WG + BEST WG

Table Top Exercise Framework

Guide to implementing a coordinated vulnerability disclosure process for open source projects

Guide to Becoming an CNA

Secure Software Development Fundamentals course LFD121

OpenSSF Project Security Baseline

Concise Guide to Evaluating Open Source Software

Concise Guide to Developing More Secure Software

Future WG work (patches welcome!)

CVD Guide for OSS Consumers

“TTX in a box”

Incident Response Plan template

IR playbooks & templates

Ways to Participate

Join a Working Group/Project

Come to a Meeting (see Public Calendar)

Collaborate on Slack

Contribute on GitHub

Become an Organizational Member

Keep up to date by subscribing to the OpenSSF Mailing List

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Thank You

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Legal Notice