Security Processes

Defining Security policies for GNU Toolchain Projects

Siddhesh Poyarekar

The What, Why, How, etc.

- What are security issues?
- Why are toolchain projects special for security issues?
- Security policy for GNU toolchain projects
- Progress and next steps

WHOAMI Now?

- Hacks on the GNU toolchain
- Argues with security researchers and CNAs on CVE assignments
- Works on Toolchain Security at Red Hat

Why are we talking about this?

- The great binutils fuzzing epidemic is starting to spread
  - Fuzzing is great, but what happens after is not
- The CVE system is broken
  - Ask and you get CVE ids, no questions asked
  - Underscore rules about software usage that get thrown out in pursuit of eyeballs
- Efforts get misdirected
  - Engineers waste time spinning backports and builds
  - Engineers waste more time trying to dispute claims
- Clearer security focus
  - Build on the frustration to build a better security posture for projects

What is a security issue anyway?

- A bug in software that allows users to do something on the computer running the software, that they otherwise wouldn't have been able to do.
- Not all security issues are equal
  - Vulnerabilities, i.e. direct compromise of availability, integrity or confidentiality
  - Missed hardening, i.e. limitations of security features
  - Flawed designs that allow commoditization of vulnerabilities
- How security issues are dealt with:
  - Vulnerabilities get CVE ids
  - Missed hardening and flawed designs get hand-wringing and occasional calls to action
    - They sometimes get CVE ids from overzealous reporters and CNAS which result in additional hand-wringing

Use Context matters

- Users interact with software through interfaces
  - Remote, via application UI (websites, web applications, etc.)
  - Locally, via user shell
  - A library API (here be context dependent dragons!)
- The context in which a bug could manifest itself matters
  - Context for a bug == scenarios in which a bug could plausibly be triggered
  - Possible != plausible
- Context defines which part of software is insecure
  - If an application uses a known insecure interface, the application needs to be fixed
  - If an interface is expected to be secure but isn't, then it needs to be fixed.
- A coir rope breaking is a bug, not a security issue
  - If it was used for, e.g. bungee jumping, the security problem is not the strength of the rope!

Security policies: setting secure use context for projects

- Define usage models for the project
  - Threat models would derive from usage models
- Define expectations of safety from different components in the project
  - One wouldn’t use a coir rope for bungee jumping
- Outline procedures to handle issues with security
- Identify contact points for security researchers

Context: GNU toolchain projects

- Not all GNU toolchain projects operate under the same context
- Not all parts of GNU toolchain projects provide safety in all contexts
- Not all programming languages provide safety to protect against arbitrary inputs

Context: runtime libraries

- Florian Weimer outlined the glibc context:
  - Security Process and Security Exceptions documents that later became $topsrcdir/SECURITY.md
- Language runtime libraries need to provide the highest usage standard (relatively speaking)
  - Even then, many limitations will apply
  - Standard definitions may put the onus on application developers vs runtime libraries
  - Libraries may warn users about safety concerns of some interfaces.
- Not every DSO is a runtime library
  - A security policy should identify runtime libraries
  - Others are just there to provide modularity, not implement a general purpose runtime.

Context: Analysis tools

- Analysis tools necessarily need to be robust on arbitrary input programs
  - Better error checking
- But analysis of arbitrary input programs MUST be in a sandbox
  - Providing sandboxing features in analysis tools is a noble goal
  - Aiming for the holy grail of allowing users to analyze arbitrary programs without sandboxing is naive and dangerous.

Context: language compilers

- Compilers can produce an output for *any* valid-ish input
- Validation limited to syntactical correctness and semantic correctness for a language
  - Safety becomes a function of the language specification
- Subjecting the compiler to arbitrary input
  - Expectation of robustness is valid
  - Expectation of maintenance of system integrity is naive
- Always sandbox when compiling arbitrary inputs
  - Compiler Explorer does it, so should you

GNU Toolchain: What could we do?

- Define a security policy
  - Researchers look for a SECURITY.*
- Play a more active role in vulnerability triage
  - CVE Numbering Authorities (CNAs) analyze vulnerabilities and assign numbers
  - Present state of triage is dismal
- Prioritize non-vulnerability security issues
  - Better diagnostics
  - Improve hardening and mitigation

GNU Toolchain: What is done and what is needed

- glibc has had a security policy for ages (Thanks Florian!)
  - Security process and exceptions wiki pages, later made into a SECURITY.md
- Binutils picked one up
  - Allowed Red Hat as CNA to somewhat stem the flow of spurious CVE assignments
- gdb
  - Is there interest?
- GCC security policy is WIP
- A glibc CNA is WIP
  - Are you a glibc contributor and have experience triaging security issues? We need you!

Thank you! Questions?

siddhesh@gotplt.org