Finding Security Vulnerabilities in Open-Source Projects

Christopher Back Kanchan Javalkar Lawson Lay Pranav Nair Adith Talupuru Rishit Viral Dr. Kevin Hamlen
Dr. Shamila Wickramasuriya

Introduction

Did you know that 75% of the cyberattacks in 2020 were at least two years old, and the oldest vulnerability found that year was 21 years old! [1] A security vulnerability is a flaw in the code of software or a system misconfiguration by which attackers can obtain direct, unsanctioned access to a network or system. Attackers could then jeopardize systems and assets using authorizations and privileges [3]. Common security vulnerabilities include cross-site scripting (XSS), SQL injection, command injection, cross-site request forgery (CSRF), etc [5].

Security vulnerabilities exist in programs anywhere, everywhere, waiting to be found. Their impact could be deterimental such as the prominent "Log4Shell" exploit or small enough to patched out in secret. We took it upon ourselves with the help of Dr. Kevin Hamlen and Dr. Shamila Wickramasuriya to find some potential security vulnerabilities of our own in open source projects through the use of their vulnerability scanner, ERLKing. ERLking specializes in Linux C programs and uses both the program's elf/bin file and source code to map out potential points of interest (POIs) or possible vulnerabilities. Through ERLking's specialization and unique approach, it produces less false positives than any other vulnerability scanner, allowing us to pin point actual vulnerabilities more accurately.

Methods

Our goal was to discover a new undocumented security vulnerability in an Open-Source Project using ERLking. Since ERLking specializes in Linux C compiled programs, we started looking for primarily C-based open source projects. Our team searched through open-source code repositories such as Github and Debian's GitLab with two ideals in mind. The project would be modestly sized with real world impact or use. CVEs (Common Vulnerabilities and Exposures), a list of publicly disclosed vulnerabilities documented by the MITRE corporation (funded by the US government), were used to narrow down the programs, as possible past vulnerabilities could lead to new ones [4]. Other leads were projects that were most likely not yet vetted extensively for vulnerabilities; e.g. lengthier source code, lower popularity, and few developers.

Our list of possible programs to scan grew but soon narrowed down due to time constraints. We experimented with the scanner, testing out different programs we found, such as photoQt, OBS, and ffmpeg. This led to us finding the one constraint we had to respect, time complexity. One downside of ERLking was the amount of time it took to generate its CPGs. Smaller programs in the size of KBs could take up to 3 hours while larger programs in MBs could take days. Even if ERLking ran inside of UTD's Computer Science Department virtual machines with a large amount of computing power and resources available for the scanner, the fact still stood.

ERLking and Scanning Vulnerabilities

ERLking is a specialized vulnerability scanner based off Joern, a source code analyzer, created by Dr. Shamila Wickramsuriya and Erick Bauman [6]. The scanner finds vulnerability through the use of both the source code and binary of the program. Generated from both source and binary are Code Property Graphs (CPGs), language-agnostic intermediate graph representation of code designed for code querying [3]. DWARF, a standardized UNIX debugging data format, is used to show the memory layout.

With the combination of both CPG and DWARF data, the graphs are used to query the memory layout of the program, searching for buffer over-writes and their affected variables. What can be queried are affected variables, Buffer overflow inducible loops (BOILs), ForConditions, Pointer null dereferences, and affected sinks. A log file with all possible detected vulnerabilities is made at the end, noted as a Point of Interest (POI).

Our focus shifted to small Debian open-source packages. Inside the large list of packages the flavour of Linux used, we narrowed down to two packages.

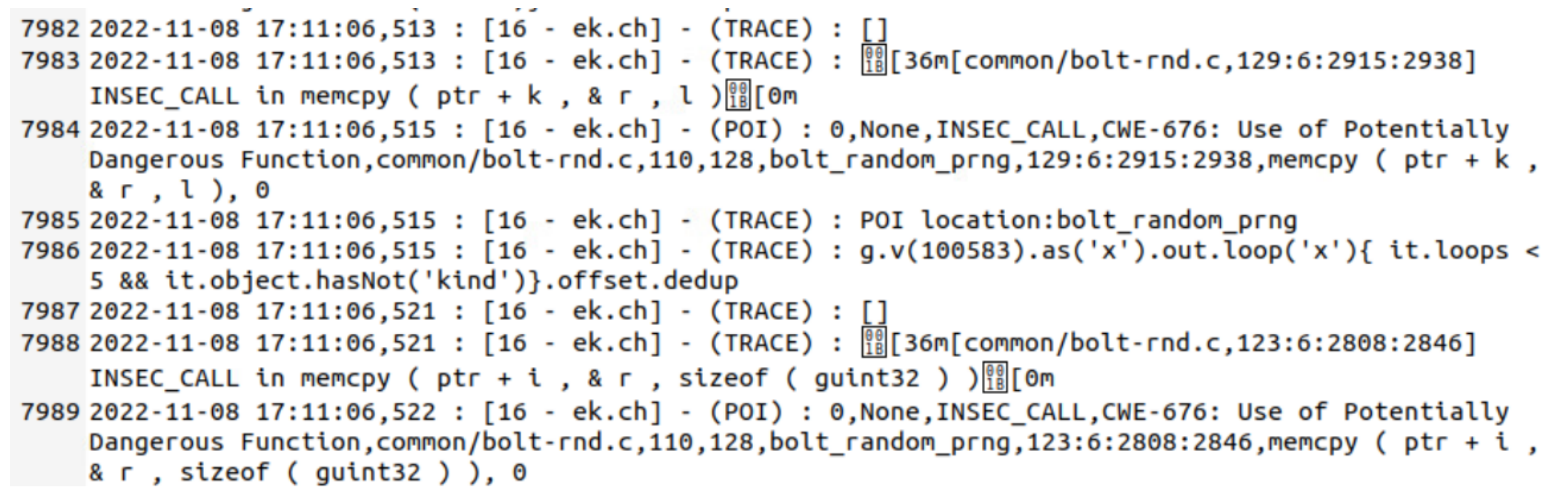
- bolt
  - The userspace system daemon responsible for enabling security levels for Intel's Thunderbolt hardware interface on GNU/Linux [8].
- cpulimit
  - A program that attempts to limit real CPU usage of a process, used to control CPU-hungry batch jobs [7].

These two programs were small enough to scan in a reasonable amount of time.

Results

After scanning bolt and cpulimit, we gathered many POIs from both programs. Most common were NULL Pointer Dereferences and second were potentially insecure calls to dangerous functions. It is important to keep in mind that POIs are not confirmed security vulnerabilities. Possible pointer dereferences could be protected against by the program, and insecure function calls could be regulated. Therefore, the only way to properly check if a POI is a proper security vulnerability, we must read the source code and see for ourselves.

Program	PTR NULL CHK	INSEC CALL	BOILs	SINKs	FOR COND
bolt	628	5	2	0	0
cpulimit	9	7	0	0	0



```
2022-11-08 17:11:06,513 : [16 - ek.ch] - (TRACE) : []
2022-11-08 17:11:06,513 : [16 - ek.ch] - (TRACE) : ␛[36m[common/bolt-rnd.c,129:6:2915:2938]
INSEC_CALL in memcpy ( ptr + k , & r , l )␛[0m
2022-11-08 17:11:06,515 : [16 - ek.ch] - (POI) : 0,None,INSEC_CALL,CWE-676: Use of Potentially
Dangerous Function,common/bolt-rnd.c,110,128,bolt_random_prng,129:6:2915:2938,memcpy ( ptr + k ,
& r , l ), 0
2022-11-08 17:11:06,515 : [16 - ek.ch] - (TRACE) : POI location:bolt_random_prng
2022-11-08 17:11:06,515 : [16 - ek.ch] - (TRACE) : g.v(100583).as('x').out.loop('x'){ it.loops <
5 && it.object.hasNot('kind')}.offset.dedup
2022-11-08 17:11:06,521 : [16 - ek.ch] - (TRACE) : []
2022-11-08 17:11:06,521 : [16 - ek.ch] - (TRACE) : ␛[36m[common/bolt-rnd.c,123:6:2808:2846]
INSEC_CALL in memcpy ( ptr + i , & r , sizeof ( guint32 ) )␛[0m
2022-11-08 17:11:06,522 : [16 - ek.ch] - (POI) : 0,None,INSEC_CALL,CWE-676: Use of Potentially
Dangerous Function,common/bolt-rnd.c,110,128,bolt_random_prng,123:6:2808:2846,memcpy ( ptr + i ,
& r , sizeof ( guint32 ) ), 0
```

ERLking's logs provides the file, function, and line number for each POI it notes down. After checking cpulimit and bolt's source code, all possible POIs were proven to be false positives.

Conclusions

Our research reveals the limitations of CPG vulnerability scanning regarding time complexity but also its effectiveness in narrowing down POIs. Using the same approach as ERLking, security researchers could use CPGs on both source and binary on a greater number of programs to narrow down false positives and spend more time verifying for actual vulnerabilities.

However, as demonstrated here, this does not mean it is guaranteed to find a vulnerability through scanning alone. Human intervention is still required for this method. To improve the chance of finding a vulnerability alone, another study with a larger set of programs could be conducted similarily. This study would need to have a longer timeframe allocated due to the increasing time complexity for both algorithm and human verification.

References

[1] E. Chickowski, "10 Stats on the State of Vulnerabilities and Exploits." https://businessinsights.bitdefender.com/10-stats-on-the-state-of-vulnerabilities-and-exploits, 2022. (accessed: 2022-11-14).

[2] J. Katsioloudes, "Today's most common security vulnerabilities explained." https://github.blog/2022-05-06-todays-most-common-security-vulnerabilities-explained/, 2022. (accessed: 2022-11-14).

[3] "Code Property Graph Specification Website | Code Property Graph Specification Website." https://cpg-spec.github.io//, 2022. (accessed: 2022-11-14).

[4] "What is a CVE?." https://www.redhat.com/en/topics/security/what-is-cve, 2022. (accessed: 2022-11-14).

[5] "What is a Security Vulnerability? | Types & Remediation | Snyk." https://snyk.io/learn/security-vulnerability-exploits-threats/, 2022. (accessed: 2022-11-14).

[6] "Overview | Joern Documentation." https://docs.joern.io/home, 2022. (accessed: 2022-11-14).

[7] "Debian – Details of package cpulimit in stretch." https://packages.debian.org/stretch/admin/cpulimit. (accessed: 2022-11-14).

[8] "Freedesktop / bolt · GitLab." https://salsa.debian.org/freedesktop-team/bolt. (accessed: 2022-11-14).