Week 7: Assess Software Security Tools

Nate Bachmeier

TIM-7241:Constructive Research Design

June 6, 2021

Northcentral University

# Assess Software Security Tools

It is incredibly challenging to build scalable, reliable, and secure software. Examining the trivial example of HelloWorld (see Figure 1) produces more defects than lines of code. First, this application fails when “þêÐŔŌ” types his name. Next, any input exceeding 256 characters crashes the program with a stack overflow. Third, nothing prevents line seven from printing arbitrary memory due to the missing null-terminating marker. Since there are 255254 valid inputs, there is a good chance that quality assurance teams will miss some of these issues.

Figure 1: HelloWorld.c

Text

Description automatically generated

## Security Tooling Categories

Software is continuously becoming more complex (Zhang et al., 2013). While this example is relatively contrived, it highlights the existing risk proportional to additional intricacy. Finding these subtle issues requires automation that examines dynamic runtime, static codebases, and package version numbers. Dynamic analysis tools like Microsoft’s Application Verifier instrument system APIs (Application Programming Interfaces)(Microsoft, 2017). This approach can detect issues like use after free and similar memory corruption issues. SonarQube transforms static code files into flow graphs to detect various defects (e.g., missing argument checks). Clair compares file versions against its vulnerability database to detect outdated references (Quary, 2021). Each of these tools specializes in one particular aspect of automated analysis.

## Potential Research Questions

Security research can be very broad or extremely narrow in scope. For instance, a collection of tools can analyze an entire AppStore. Casting this wide net enables population analysis (e.g., frequency of critical issues). Alternatively, the project can focus on a single application or technology to hunt zero-day vulnerabilities.

# Using Clair

Amazon Web Services (AWS) integrates Clair into its Amazon Elastic Container Register (ECR) service (see Figure 2). The feature runs automatically after receiving an updated container version. This specific report instance identifies three critical issues and demonstrates the tool’s value. When administrators can effortlessly gain these insights, it promotes security and reliability across their infrastructure.

Figure 2: Amazon ECR Vulnerability Analysis

Graphical user interface

Description automatically generated

## SCP Vulnerability

Secure Copy (scp) is a standard utility for transferring data between Linux environments. This particular version is vulnerable to the caller modifying the search path (Suse, 2020). When attackers change this value, it distorts relative to absolute path resolutions. That behavior can cause the wrong binary to execute, compromising the system’s integrity.

## F2FS Vulnerability

Next, the Flash-Friendly File Systems (F2FS) is a purpose-built storage layer for NAND flash memory devices (Kernel, n.d.). While removing dirty page segments, the implementation relies on a correctly set enumeration flag set. Malicious or erroneous drivers that modify those flags can trigger an out-of-bounds write request (BobFuzzer, 2019). Typically, this behavior introduces a Kernel Panic and forces a reboot. However, a careful sequence of events could overwrite a crucial operating system data structure. This extreme situation compromises the system’s security and integrity.

## MariaDB Vulnerability

The third issue states that the MariaDB service contains a command execution vulnerability. Triggering the issue requires crafting a malicious search path that feeds into an evaluation (eval) function (Mitre, 2021). Those commands execute as the MariaDB’s Linux account and permission set. Attackers can leverage those credentials to modify the local machine or circumvent network firewalls. However, modifying the search path requires SUPER user access to the database. Administrators need to restrict that account (e.g., strong password) and avoid using it wherever possible (e.g., front-end webserver).

# Conclusions

Even trivial software packages contain vulnerabilities. Software developers need to find these issues through sophisticated tooling like Clair. AWS automatically surfaces some of those defects through reporting capabilities. Those reports provide categorical findings counts and quantify its effectiveness. Administrators can also follow the report links for instructions on upgrading the impacted libraries. This workflow provides the visibility and prioritization necessary to keep the infrastructure secure.

Researchers can also use these tools to study app populations (e.g., Google PlayStore). Numerous statistical questions arise from those investigations. For instance, how many vulnerabilities does the typical app contain? Those discoveries influence training programs and future API designs. That ultimately makes the ecosystem more secure, reliable, and developer-friendly.

# References

BobFuzzer. (2019, December 27). *CVE-2019-19814*. Retrieved from GitHub: https://github.com/bobfuzzer/CVE/tree/master/CVE-2019-19814

Kernel. (n.d.). *What is Flash-Friendly File System*? Retrieved from Kernel: https://www.kernel.org/doc/html/latest/filesystems/f2fs.html

Microsoft. (2017, April 20). *Application Verifier*. Retrieved from Microsoft: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/application-verifier

Mitre. (2021, March 3). *CVE-2021-27928*. Retrieved from CVE Database: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-27928

Quary. (2021). *Clair*. Retrieved from GitHub: https://github.com/quay/clair

SonarQube. (2021). *What is SonarQube*? Retrieved from SonarQube: https://www.sonarqube.org/

Suse. (2020, June 18). *Bug 1158095 - (CVE-2019-14889) VUL-0: CVE-2019-14889: libssh: unsanitized location in scp could lead to unwanted command execution*. Retrieved from Bugzilla: https://bugzilla.suse.com/show\_bug.cgi?id=CVE-2019-14889

Zhang, J., Sagar, S., & Shihab, E. (2013). The evolution of mobile apps. *International Workshop on Software Development Lifecycle for Mobile* (pp. 1-8). New York, NY, USA: Association for Computing Machinery. doi:10.1145/2501553.2501554