Software Assurance Tips

A product of the Software Assurance Tips Team[2]

Jon Hood

Monday 21st June, 2021

1 Living off the Land

Updated Wednesday 11th February, 2026

Living off the Land (LotL) refers to using the tools, scripts, libraries, and binaries that are native and pre-installed with an environment.[6, p. 299] Attackers use the existing tools of their exploited environment to decrease their footprint and avoid detection, often blending in by using legitimate tools and administrative functions in a malicious way.[1, p. 45] Nevertheless, developers can use LotL concepts to decrease the attackable footprint of their software as a mitigation strategy against attacks as well!

1.1 LotL: An Attacker's Perspective

When attacking an environment, penetration testing teams often attempt to take an inventory of what is available to see what can be utilized to establish persistence and pivot in the environment without detection. If the attacker has to install specialized tools and packages, intrusion detection and prevention systems can be configured to look for those packages as part of a defense-in-depth mitigation strategy.

1.2 LotL: A Defender's Perspective

Defensive (blue) teams should be familiar with the attacks and threats posed by penetration testing (red) teams. In 2013, the DoD used the following definition of Software Assurance: "the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software, throughout the life cycle."[5, p. 255 §933.e.2] In 2017, the Office for the Secretary of Defense clarified the definition of "functions as intended" to mean "*only* functions as intended" by showing confidently that the software meets functionality and test coverage requirements and also "does not perform unrequired functions."[4, p. 7] Since 2017, the caveat "and only as intended" has been codified as part of the DoD's definition of Software Assurance.[3, p. 2]

Developers are often tempted by heavyweight frameworks that provide sundry capabilities. For low-assurance tasks, these frameworks permit rapid development and deployment of solutions. In tactical environments, these bloated dependencies are much more difficult to maintain. Large frameworks must be maintained and reviewed for CVEs and security updates in their deployed environments, even when security findings affect areas of the framework that are not utilized by the solution.

1.3 LotL In Practice

Recently, one particular application provided a framework that used libzip, zlib, 7-Zip, and the System.IO.Compression ZipArchive routines for reading in compressed files at different parts of the application. During the Software Assurance scans, we were able to create a payload using the Zip Slip vulnerability to upload and execute any payload we needed. During postmortem analysis, the developers cleaned up the vulnerability that was used, but forgot to verify that another location of the code used a vulnerable version of 7-Zip that could stage the same payload!

1.4 Mitigations

Mitigations to attackers that attempt LotL techniques involve making sure that the land an attacker would get is insufficient for staging more advanced attacks. When an attacker only gets access to a minimalist, sandboxed island, there isn't enough land to live on. They must either be noisy, terraforming the land into a new environment and risking detection, or silently move on to another, more vulnerable environment to avoid detection.

Some additional information to consider:

- If installing a webserver, determine the languages web applications are written in. If there are no perl, PHP, Ruby, or .NET applications, then why install those frameworks to begin with?
- Watch for additional tools being installed. Why should the sandboxed web account have access to netcat, telnet, or the entire suite of tools? Why is there a compiler installed on the server?
- If a service does not need to be touched by a user, does it make sense to give that service its own sandbox and account with access only to the application that needs it to run?

A defender should follow the least-privilege and minimum functionality practices when developing software. By refusing to give an attacker much land to live on, developers reduce their exposure space and discourage malicious activities. As a byproduct of using minimal frameworks and dependencies, maintenance becomes easier, particularly on disconnected and tactical systems.

Containers that provide only the needed functionality should be preferred over monolithic containers that provide entire operating systems and environments. By sandboxing user execution, an attacker that successfully gets a foothold into an application is rewarded with their very own tiny sandbox to play in, while non-containerized, multi-user execution often provides attackers with the keys to the entire kingdom. Compromise is not a fun topic to consider, but risk prevention is not a binary, pwned-or-not decision. I'd prefer that my software not be compromised, but if it should become impaired, I'd rather the malicious attacker only be able to steal the sandbox rather than the entire kingdom!

If you give a crook your sandbox, they'll defecate in your sand. But if you give a crook your kingdom, they'll usurp your entire domain.

References

[1] Alan Calder. The Cyber Security Handbook. Prepare for, respond to and recover from cyber attacks. IT Governance Publishing Ltd, 2020. ISBN: 978-1-78778-261-7.

[2] Jon Hood, ed. SwATips. https://www.SwATips.com/.

[3] Thomas Hurt. Achieving DoD Software Assurance (SwA). Tech. rep. Springfield, VA: 20^{th} Annual NDIA Systems Engineering Conference, Oct. 2017. URL: https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2017/systems/Thursday/Track1/19911_Hurt.pdf.

[4] John R. Marien and Robert A. Martin. Incorporating Software Assurance into Department of Defense Acquisitio Tech. rep. Department of Defense (DoD) Software Assurance (SwA) Community of Practice (CoP) Contract Language Working Group, 2017. URL: https://rt.cto.mil/wp-content/uploads/2019/06/Incorporating-SwA-Contracts-2017-11-15.pdf.

[5] NATIONAL DEFENSE AUTHORIZATION ACT FOR FISCAL YEAR 2013. Public Law 112–239. Jan. 2013. URL: https://www.congress.gov/112/plaws/publ239/PLAW-112publ239.pdf.

[6] Tim Rains. Cybersecurity Threats, Malware Trends, and Strategies. Learn to mitigate exploits, malware, phish Packt Publishing, 2020. ISBN: 978-1-80020-589-5.