Enhancing Software Supply Chain Security

Prepared For:
National Institute of Standards and Technology (NIST)

Position On:
Guidelines for software integrity chains and provenance,
EO Sections 4(e)(ii, vi, and viii).

Prepared By:
Michael Hanchak, MongoDB

May 26, 2021

Vendors can assist customers in their security program by documenting expected product behaviors and key security information in a programmatically ingestible structured format. A repeated adage in security is “the defender can only defend against known attacks; attackers can probe for unknown vulnerabilities.” One way to help address this asymmetry is to enable defenders to quickly identify and classify unknown behavior as threat. A detailed and accurate behavior baseline enables customer security teams to create relatively high confidence security detections for abnormal system events. Furthermore, a strong vendor and customer partnership in this area can serve as a potential end to end test of the vendor’s security.

Clearly documenting product behavior has the ability to simultaneously improve detection coverage and increase operational efficiency for customers. A proven threat detection strategy is identifying deviations from expected behavior. Techniques like machine learning or comparing behavior across multiple instances do exist to establish a baseline. Unfortunately these are not always as accurate, resource efficient, or readily accessible to security teams. Conversely, time spent trying to distinguish benign behavior from threat delays response. It also acts as a tax preventing security programs from investing elsewhere.

The following are just a few examples of behaviors that may be useful to clearly communicate in a consistent structured format such as JSON:

- Known, trusted third-party systems with which the product communicates
    - Addresses (domain or IP ranges)
    - Method (TLS, SSH, etc)
    - Feature or reason - ie. automatic updates

- Customer systems with which the product communities
    - System type - ie. LDAP directory, other cluster node, etc
    - Method and port - ie. LDAPS, TLS over port 8443, etc
    - Privileges required
        - Minimum required for base operation
        - Mapped to feature and documentation when optional
    - Feature or reason - ie. account federation or config management
- Patterns of system commands executed mapped to feature - ie. exec check_for_update.sh to power an automatic update feature
- Default accounts
- Software signing keys
- Software hashes

Clearly outlining the information above can power the following example detections:

- Communication to an unknown domain indicating potential data exfiltration or command & control
- Accessing unexpected internal systems indicating potential lateral movement
- Invoking new system commands indicating potential escalation of privilege
- Usage of a builtin default account credential indicating potential initial compromise
- Execution of software with an unknown hash indicating potential compromise or persistence
- Execution of software with an unknown certificate signing authority indicating potential compromise or persistence

Aligning to a consistent structured format across vendors encourages customer teams to invest in automation. Many of these detections can be templated in advance and reused across products, including both on-premise and hosted offerings like SaaS and PaaS. Furthermore, the support and backing of an agency like NIST might encourage some security vendors to embrace the standard. This can potentially be an accessible method for security teams to quickly tune endpoint detection and response (EDR), security incident and event management (SIEM), or intrusion detection systems (IDS).