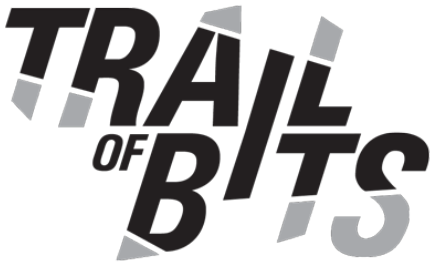

Linux Kernel Release Signing

Security Assessment

April 5, 2021

Prepared For:
Derek Zimmer | *Open Source Technology Improvement Fund*
derek@ostif.org

Greg Kroah-Hartman | *Linux Foundation*
gregkh@linuxfoundation.org

Konstantin Ryabitsev | *Linux Foundation*
konstantin@linuxfoundation.org

Prepared By:
Jim Miller | *Trail of Bits*
james.miller@trailofbits.com

Mike Martel | *Trail of Bits*
mike.martel@trailofbits.com

Opal Wright | *Trail of Bits*
opal.wright@trailofbits.com

Executive Summary

On behalf of the Linux Foundation, the Open Source Technology Improvement Fund (OSTIF) sought a third-party review of the Linux kernel release signing process. From March 29 to April 2, 2021, Trail of Bits reviewed the security of this process and provided recommendations for improvements. Trail of Bits conducted this assessment over two person-weeks, with two engineers reviewing the kernel release process as described by the Linux Foundation.

Trail of Bits began the assessment by reviewing the provided documentation on the Linux kernel release signing process. This led to a technical discussion with representatives of the Linux Foundation pertaining to their design. After strengthening our understanding of the overall process, we began to analyze the design for any weaknesses or opportunities for Defense-in-Depth recommendations.

To analyze the security of the kernel release signing process, Trail of Bits enumerated various threat scenarios against the system, which are detailed in this report. We were then able to identify issues and develop recommendations for improving the system such that it would be secure if any of those threats materialized. We detail seven issues, working off of six threat scenarios.

The process reviewed by Trail of Bits was designed primarily in response to an attack on kernel.org in 2011. Its goal is minimizing the amount of trust required in each infrastructure component of the system. We found this current design achieves this goal to a degree; however, the recommendations in this report should help reduce the amount of implicit trust placed in the current design's infrastructure. For instance, issue TOB-LFKS-006 describes how adding more external validation could increase the robustness of the system and reduce this implicit trust.

We would encourage the Linux Foundation to consider all of our recommendations and to improve the documentation of its process. The documentation supplied to Trail of Bits was informative but outdated. A current, comprehensive description of the process would make it easier to enforce compliance and identify any weaknesses.

Project Dashboard

Application Summary

Name	Linux Foundation
Version	Kernel Release Signing
Type	Process Design
Platforms	Linux

Engagement Summary

Dates	March 29 – April 2, 2021
Consultants Engaged	2
Level of Effort	2 person-weeks

Vulnerability Summary

Total High-Severity Issues	0	
Total Medium-Severity Issues	1	■
Total Low-Severity Issues	4	■ ■ ■ ■
Total Informational-Severity Issues	2	■ ■
Total	7	

Category Breakdown

Access Controls	2	■ ■
Auditing and Logging	2	■ ■
Authentication	2	■ ■
Cryptography	1	■
Total	7	

Engagement Goals and Coverage

The engagement was scoped to provide a security assessment of the Linux kernel release signing process. We performed the assessment by reviewing the documentation supplied by the Linux Foundation and engaging in technical discussions with Linux Foundation representatives. We identified several possible threat scenarios (detailed in the "Threat Scenarios" section) and developed recommendations for mitigating those scenarios.

Specifically, we sought to answer the following questions:

- Are GPG keys managed safely?
- Are releases signed and verified safely?
- Does the design use any weak cryptographic primitives?
- Is the process susceptible to any known cryptographic attacks?
- Does the security of the release process unnecessarily rely on the security of infrastructure not trusted by the Linux Foundation?

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **Update all documentation related to the release process to accurately reflect the latest version of the process.** This will make it easier to ensure compliance and identify weaknesses.

❑ **Require individuals with access to significant repositories or systems to use a smart card device to store sensitive key material.** TOB-LFKS-001

❑ **Consider mandating the use of smart card devices that require physical touch to validate each smart card operation.** If that is not possible, add guidance recommending that a smart card be physically connected to a workstation only when it is required to complete an operation. TOB-LFKS-002

❑ **Work with administrators and developers to document current procedures and policies and compile that information into a single set of documents that can be updated as necessary.** TOB-LFKS-003

❑ **Identify effective ways to widely advertise developers' key fingerprints.** These could include adding key fingerprints to email signatures, periodically posting them in mailing lists, or referencing them in conference presentations. TOB-LFKS-004

❑ **Choose a single algorithm and key size for new keys incorporated into the kernel web of trust and the PGP key repository.** Trail of Bits recommends using elliptic curve algorithms; GnuPG supports ECDSA and Ed25519 signatures starting from version 2.2. TOB-LFKS-005

❑ **Consider releasing tooling that can compare release tarball content with the content of the tagged Git release, as well as tooling that can ensure that all commits to key repositories hosted on kernel.org are signed with an expected identity.** Also consider running a verifier on kernel.org systems. TOB-LFKS-006

❑ **Develop an appropriate key rotation schedule to limit the impact of a compromised SSH key.** TOB-LFKS-007

Long Term

❑ **As the release process evolves over time, keep all of the documentation up-to-date with the latest version.** This will make it easier to ensure compliance and identify weaknesses.

❑ **Periodically review policies and procedures, assessing their applicability and appropriateness.** Update the documentation as policies and procedures change. TOB-LFKS-003

❑ **Continue advertising keys through multiple channels, and work with partners to provide easily-found sources of public-key corroboration.** TOB-LFKS-004

❑ **Work with developers to generate new, standard-compliant keys and integrate them into the kernel web of trust.** TOB-LFKS-005

❑ **Consider advocating for interested independent parties to run these verification tools to bolster the integrity verification mechanisms of the wider Linux kernel community.** TOB-LFKS-006

❑ **Ensure that key rotation policy is followed and that administrators are practiced in key rotation procedures.** This will limit the threat posed by compromised keys and ensure that administrators are capable of quickly rotating keys when a compromise is discovered. TOB-LFKS-007

Threat Scenarios

This section describes the scenarios that guided our analysis and subsequent development of recommendations. These scenarios are not meant to be exhaustive but illustrate how a sufficiently advanced external adversary could compromise the integrity of the kernel signing and release process.

Scenario 1: Compromise of kernel maintainer workstation

Possible attack vectors
Web browser exploitation, malicious email attachments, coercion, and physical access

Impact
Kernel maintainer systems include GPG and SSH keys to authenticate and sign releases. If a maintainer did not use a smart card to store sensitive key material, an attacker could retrieve the key material, intercept the passphrase entry, and (for example) push malicious code to repositories and subsequently release a malicious kernel with a valid developer signature. An attacker could also use sensitive key material to move laterally into kernel.org systems to carry out certain of the scenarios described below.

Scenario 2: Compromise of kernel.org administrator workstation

Possible attack vectors
Web browser exploitation, malicious email attachments, coercion, and physical access

Impact
This scenario is similar to scenario 1, but the use of smart cards to store key material would complicate lateral movement. However, if an attacker intercepted GPG passphrases, the attacker would be able to use the key material on the smart card if the card did not support functionalities like requiring physical touch to be triggered. If the administrator had already connected to the Wireguard VPN, the attacker could piggyback on this connection to access kernel.org back-end systems and move laterally, potentially facilitating other threat scenarios.

Scenario 3: Compromise of kernel.org front-end systems

Possible attack vectors
Prior compromise such as that in scenario 2, remote exploitation of an exposed service, and physical access

Impact

If an attacker were able to manipulate front-end systems, the attacker could potentially modify information displayed on kernel.org or the Git front end or change keys delivered to users employing WKD with GPG. This could allow the attacker to upload malicious kernel releases or insert additional signing keys associated with trusted identifiers, which could be used as part of a more complex attack (such as those described below). In the event that an attacker gained exclusive access to front-end systems, though, measures such as the `autosigner` mechanism would prevent the attacker from executing long-term subversion of kernel releases without detection.

Scenario 4: Compromise of kernel.org git master server

Possible attack vectors
Prior compromise such as that in scenario 1 or 2, remote exploitation of an exposed service, and physical access

Impact
An attacker who has persistent access to the `git master` server could modify `gitolite` and `git` configurations, which could enable the attacker to make commits as other valid users (e.g., pushing unsigned commits as Linus). Without sufficient commit-monitoring measures, malicious code could be inadvertently included in future kernel releases signed with a valid key. If an attacker also possessed key material valid for signing a release, the attacker could push a `git tag` to kick off a release process with malicious code as well.

Scenario 5: Compromise of kernel.org kup-server

Possible attack vectors
Prior compromise such as that in scenario 1 or 2, remote exploitation of an exposed service, and physical access

Impact
An attacker with persistent access could modify the processes running on `kup-server` to subvert the upload of a valid tarball signature for a kernel release. For instance, an attacker could replace the tarball and signature file that is output by `kup` and sent to temporary storage before the `marshall` server synchronized the data. As the GPG signature is not verified again, `sha256sums.asc` generation would include the malicious tarball and signature, and they would be pushed to mirrors. If the tarball were signed without a valid developer identity, this type of subversion would likely be quickly detected by the community, as signature verification would fail. However, if the attacker also had front-end access, like in scenario 3, or possessed valid key material of a maintainer, like in scenario 1, detection would be more difficult.

Scenario 6: Compromise of kernel.org marshall server

Possible attack vectors
Prior compromise such as that in scenario 2, 4, or 5

Impact
This scenario is similar to scenario 5. However, an advanced attacker could conceivably modify or influence the operation of `grokmirror` as well as the content of files stored on NFS to push malicious tarballs or signatures and modify the content of `pub/scm`. Unlike scenario 5, in which an attacker would need to hijack an ongoing release process to insert a malicious tarball, this scenario could allow an attacker modifying the content synchronized via `grokmirror` to trigger a release. The community would likely detect such a release, as valid `git` tags would be missing from the authoritative repositories.

Findings Summary

#	Title	Type	Severity
1	Use of smart card for GPG and SSH not enforced for key individuals	Access Controls	Medium
2	Recommended smart card does not require touch activation	Access Controls	Low
3	Lack of documented key management policies and procedures	Auditing and Logging	Low
4	Lack of public-key corroboration resources	Authentication	Informational
5	Insecure or invalid keys are allowed within the web of trust	Cryptography	Informational
6	Lack of external integrity validation mechanisms	Auditing and Logging	Low
7	Lack of SSH key rotation	Authentication	Low

1. Use of smart cards for GPG and SSH not enforced for key individuals

Severity: Medium
Type: Access Controls
Threat Scenario: Scenario 1

Difficulty: Medium
Finding ID: TOB-LFKS-001

Description
To verify kernel updates, Linux kernel developers produce signed tags in their git trees and produce a signature over the entire tarball for a new release using their GPG keys. These updates are then delivered to various components of the kernel.org infrastructure using SSH sessions.

Individuals with commit rights on key Linux kernel repositories are not required to store private key material used for GPG or SSH on a separate smart card device, such as a Nitrokey or Yubikey.

Exploit Scenario
Alice is a Linux kernel maintainer who stores private key material on a user-accessible block device. Eve, an attacker, is able to install malware on Alice's workstation. Eve is able to exfiltrate private key material from Alice's workstation and could attempt to brute-force passphrases or to install a keylogger to record passphrase entry. Eve could then create valid signatures and authenticate to some kernel.org services using the stolen key material.

Recommendation
Short term, require individuals with access to significant repositories or systems to use a smart card device to store sensitive key material. If that is not a viable option, consider using an alternative mechanism, such as a TPM, to protect sensitive cryptographic material.

2. Recommended smart card does not require touch activation

Severity: Low
Type: Access Controls
Threat Scenario: Scenario 1, 2

Difficulty: Medium
Finding ID: TOB-LFKS-002

Description
The Linux Foundation recommends that kernel developers use smart cards, specifically Nitrokeys, to secure their private key material. Linux Foundation-issued Nitrokeys do not require users to perform any physical actions when using smart card functions. Other devices can be configured to require the user to touch the device before the smart card operations occur. As a result, the Nitrokey is protected only by a passphrase while inserted into a workstation.

While touch activation does not prevent all classes of attacks, such as ones that replace binaries on disk (e.g., for GPG and SSH) or leverage session hijacking, it prevents entire classes of less sophisticated attacks and improves the security posture of a given end-user.

Exploit Scenario
Alice is a Linux kernel maintainer who stores private key material on a Nitrokey. Eve, an attacker, is able to install malware on Alice's workstation. Eve is able to intercept the entry of Alice's passphrase while Alice is using the Nitrokey. Without Alice's knowledge, Eve can make requests to the Nitrokey while it remains plugged in to the device and subsequently authenticate as Alice and access or modify sensitive systems and repositories.

Recommendation
Short term, consider mandating the use of smart card devices that require physical touch to validate each smart card operation. If that is not possible, add guidance recommending that a smart card be physically connected to a workstation only when it is required to complete an operation, which would help prevent an attacker from using an attached smart card device without its user's knowledge.

3. Lack of documented key management policies and procedures

Severity: Low
Type: Auditing and Logging
Threat Scenario: N/A

Difficulty: N/A
Finding ID: TOB-LFKS-003

Description
There is no centralized, authoritative documentation laying out policies and procedures for key revocation, generation, or rotation or other key management tasks. Without such documentation, users and administrators are more likely to make serious errors when engaging in routine and emergent key management tasks.

Recommendation
Short term, work with administrators and developers to document current procedures and policies and compile that information into a single set of documents that can be updated as necessary.

Long term, periodically review policies and procedures, assessing their applicability and appropriateness. Update the documentation as policies and procedures change.

4. Lack of public-key authentication resources

Severity: Informational
Type: Authentication
Threat Scenario: Scenarios 3 and 4

Difficulty: N/A
Finding ID: TOB-LFKS-004

Description
To verify the content of kernel updates, each commit in the git tree produces a signed tag, and each release is accompanied by a signature over the release's tarball. Public keys for Linux developers, as well as the associated key signatures forming the web of trust, are managed from a single location. Compromise of the git.kernel.org server would allow an attacker to provide users with a web of public keys not controlled by kernel developers, enabling them to post malicious kernel "releases" that would validate against the attacker's public keys.

Bootstrapping trust for public-key systems is a hard problem, and is certainly not unique to the Linux kernel. Any software that relies on digital signatures for verification must first "bootstrap" trust by ensuring that users have the correct public key. It creates a circular problem that is difficult to solve in the general case.

The Linux Foundation is uniquely equipped to alleviate this problem. Because of Linux's community and commercial support, kernel developers have many ways to distribute PGP key fingerprints for important developers. Key fingerprints can be included in conference presentations, periodically published on news sites such as lwn.net, included in email signatures, or published on websites maintained by Linux Foundation partners like Red Hat or IBM. Key fingerprints hosted on the kernel.org infrastructure could then be validated against multiple public sources, reducing the likelihood of a malicious key being trusted.

Recommendation
Short term, identify effective ways to widely advertise developers' key fingerprints. These could include adding key fingerprints to email signatures, periodically posting them in mailing lists, or referencing them in conference presentations.

Long term, continue advertising keys through multiple channels, and work with partners to provide accessible sources of public-key corroboration.

5. Use of older public-key algorithms and standards within web of trust

Severity: Informational
Type: Cryptography
Threat Scenario: N/A

Difficulty: High
Finding ID: TOB-LFKS-005

Description

PGP keys used by kernel developers vary significantly in terms of algorithm and key size. RSA is the most commonly used algorithm, followed by DSA and elliptic curve algorithms. Work estimates for attacking algorithms based on integer factorization and integer discrete logarithms vary widely, and the algorithms are frequently subject to subtle new failure modes. Trail of Bits generally recommends moving away from RSA where possible in favor of elliptic curve algorithms.

Since these keys are used to verify code that is eventually incorporated into the kernel, modern primitives should be used. Using a single modern algorithm and key size will help reduce the attack surface for sophisticated attackers.

Recommendation

Short term, choose a single algorithm and key size for new keys incorporated into the kernel web of trust and the PGP key repository. The current kernel developer guidance suggests using ECDSA or Ed25519 keys. Requiring all new keys to conform to this guidance would be an effective step toward standardization.

Long term, work with developers to gradually replace older RSA and traditional DSA keys with new policy-compliant keys and integrate them into the kernel web of trust. Establish a “sunset” date by which all keys should be switched over.

6. Lack of external integrity validation mechanisms

Severity: Low
Type: Auditing and Logging
Threat Scenario: Scenarios 3, 4, 5, and 6

Difficulty: N/A
Finding ID: TOB-LFKS-006

Description
Kernel releases involve a series of steps such as merging changes in Git repositories, pushing tags, and generating a tarball for release. Currently, verification of the steps' integrity largely depends on the wider community to notice incorrect or malicious behavior. Although this can be effective, additional integrity checks would greatly increase the robustness of the system and help reduce the implicit trust placed in the infrastructure.

Recommendation
Short term, consider releasing tooling that can compare release tarball content with the content of the tagged Git release, as well as tooling that can ensure that all commits to key repositories hosted on kernel.org are signed with an expected identity. Also consider running a verifier on kernel.org systems.

Long term, consider advocating for interested independent parties to run these verification tools to bolster the integrity verification mechanisms of the wider Linux kernel community.

7. Lack of SSH key rotation

Severity: Low
Type: Authentication
Threat Scenario: Scenario 1

Difficulty: High
Finding ID: TOB-LFKS-007

Description
Currently, SSH keys used to access kernel.org infrastructure are static. Because SSH keys can often be leveraged to access additional systems, they are frequently targeted by attackers. Under the current setup, recovery of a single developer's SSH key could allow indefinite access to kernel.org resources.

A key rotation schedule would mitigate the impact of an SSH key compromise on the kernel.org system. Moreover, with a system in place for rotating SSH keys, the Linux Foundation could respond to an attack that compromises these keys more quickly.

Recommendation
Short term, develop an appropriate key rotation schedule to limit the impact of a compromised SSH key.

Long term, ensure that key rotation policy is followed and that administrators are practiced in key rotation procedures. This will limit the threat posed by compromised keys and ensure that administrators are capable of quickly rotating keys when a compromise is discovered.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.

Medium	Individual users' information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	Commonly exploited public tools exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.