System Transparency

5th revision, July 2019, by Fredrik Strömberg (stromberg@mullvad.net)

Introduction

Computer security exists to facilitate trust, and a fundamental part of security is system integrity. Unfortunately, computer systems are rife with vulnerabilities that can be used to compromise them, and the endless stream of security patches serve both as immunization against as well as recipes for exploitation. System administrators are therefore in recurring races to patch before attackers exploit.

Even with a good patching story, a system, once compromised, is often costly to get back to a secure state. A lack of trustworthy audit trails may prevent discovering the initial cause of the breach resulting in inadequate mitigations. Regrettably, modern systems also offer many places to hide and gain malware persistence. The technical challenges involved in maintaining system integrity with a relatively high assurance prevent most organizations from doing so.

The following established concepts have security properties that are orthogonal to one another, each able to assist in ensuring and maintaining integrity. Through their systematic composition, new security properties emerge that are the subject of this paper.

A **key ceremony** is a procedure where a key pair is generated for later use as a signing key. It may take place in a vault and involve multiple witnesses, a log book of events, a video recording, and so on.

Physical write-protection of firmware ensures integrity of the first instructions executed after a hard reset.

Tamper detection is used to trigger defensive measures if a security boundary, such as a server casing, is breached.

Reproducible builds is the process of deterministically compiling source code in a way that ensures a given source code always results in the same bit-identical artifact.

Measured and verified boot are security features that measure and verify parts of the boot chain respectively, often through the use of a TPM. Boot chain measurements may then be used by the TPM to provide **remote attestation**.

Immutable infrastructure is an approach to system administration that advocates rebuilding and reprovisioning system images rather than modifying already running systems using, for instance, SSH. Once deployed, an immutable system is never modified, it is only replaced.

Certificate Transparency is a framework intended to monitor and audit the issuance and existence of SSL/TLS certificates. It consists of a signed and publicly auditable append-only log, a log monitor, and a log auditor.

Through the systematic composition of these concepts, we now introduce System Transparency – a novel design approach for computer systems intended to offer deterrence, prevention, and detection of attacks by combining a provisioning ritual, write-protected firmware, tamper detection, reproducible builds, remote attestation, immutable infrastructure, and a signed and auditable append-only log. Used correctly, System Transparency will prevent malware persistence, provide an extensive and trustworthy audit trail, and eventually self-heal after compromise. Within certain limitations it can be used to prove to the owner, system administrator, user, or a third party exactly what is currently running on the system and what it has been permitted to run in the past.

System Transparency

System Transparency begins with a provisioning ritual similar to a key ceremony, in the sense that it is a meticulously executed procedure with an audit trail. During this ritual, the boot ROM on the target platform is reprogrammed with firmware containing hardware initialization code as well as public keys for verification of the next boot stage. At least one key should belong to the system owner, and at least one key should belong to a signed append-only log similar to a Certificate Transparency Log. Ideally the log is run by an organization independent from the system owner.

The source code for the firmware, as well as all other artifacts executed by the platform, must be available to parties auditing the running system. Furthermore, all artifacts must be reproducibly built, as otherwise a strong link wouldn’t exist between source code and artifact.

The goal of the provisioning ritual is to convince future auditors that the stated hardware specifications are correct; that the boot ROM was programmed with an artifact with a specific checksum; and, finally, to tie the platform to a newly generated public key contained in the platform TPM. Assurance that the platform has not been tampered with after the provisioning ritual is provided by tamper detection switches connected to the casing and TPM; through the use of an enclosure PUF; or similar measures.

After a hard reset, the platform initializes hardware and proceeds to verify the next boot stage. Verification of the next boot stage by using one or more keys controlled by the system owner ensures that only system images approved by the owner can run. Verification by using at least one key only used in a transparency log context ensures that only artifacts that have been submitted to the transparency log will run on the platform. Artifacts might be submitted directly to the transparency log or indirectly by submitting its checksum. As a proof-of-concept, one could piggyback on existing Certificate Transparency Logs by registering a certificate with a Common Name of e.g. checksum.transparency.your-domain.tld.

In order to offer correct remote attestation, each stage of the boot chain must be measured into the TPM before execution. Each stage that needs to load further stages must also verify those stages using keys from the system owner as well as keys used by a transparency log.

Finally, System Transparency requires that the system executes artifacts that are immutable in the sense that they do not offer interactive arbitrary access. Crucially, it must not offer an avenue to change system behavior in ways that can not be predicted by inspection of artifacts in the transparency log. For instance, if a system image offers SSH access, a malicious system owner might change data processing in ways that break the privacy policy it has marketed to its users. A system correctly designed for System Transparency would require the system owner to create a new artifact containing the malicious modifications, submit it to the append-only transparency log, and finally deploy it. Such deployment requirements might also deter third parties from applying legal pressure or otherwise force an organization to compromise system integrity and thereby the security and privacy of its users. With sufficient tooling and auditing of the transparency log, strange updates would inevitably face scrutiny.

Conclusion

We have described System Transparency – a novel design approach to ensure and maintain system integrity which offers emergent security properties through systematic composition of orthogonal security technologies. It facilitates trust in the hardware and initial state of the system through a provisioning ritual and tamper detection which together with a TPM and firmware write-protection establishes the root-of-trust as well as prevents malware persistence.

The requirements of reproducible builds in combination with immutable infrastructure deter and prevent malicious modification during the build stage as well as during runtime. The requirements of remote attestation of the boot chain in combination with a transparency log provide assurances of the current system configuration, as well as an audit trail of previous configurations.

Finally, when a platform using System Transparency is compromised due to an unpatched application, it can simply reboot, load an updated system image, and attest its new, patched, and uncompromised boot chain to its system administrator or users.

A proof-of-concept implementation is in progress.

Future Work

System integrity starts with the hardware and continues with the first instruction. Instead of keeping boot ROM and TPM separate, they might be combined into one for higher assurance of system integrity, provided the right components are used.

System Transparency as a security architecture results in the entire system becoming a “white box” from a penetration testing approach. Assuming the system is intended to be publicly auditable, attackers will know exactly which software is running, which versions, and its configurations. The impact of this should be investigated further.

The details of the transparency log need to be figured out. Binary Transparency exists as concepts in various flavors, but append-only logs in the context of System Transparency are slightly different.

Acknowledgements & Inspiration

This paper would not have come about were it not for the hard work and writings of others. Thank you to Joanna Rutkowska for writing the stateless laptop paper. Thank you to Leslie Lamport for your papers on state machines. Thank you to langsec.org for your writings on input handling. Thank you to Trammell Hudson for the Heads firmware, and your talks on remote attestation and more resilient systems. Thank you to Ron Minnich for leading the way with coreboot and LinuxBoot.

Thank you to all of my Mullvad team colleagues without whom I would neither have had the inspiration nor the opportunity to start working on this project. Thank you to Linus Nordberg and Peter Stuge who both acted as sounding boards and provided useful insights and feedback throughout the process. Thank you to Daniel Berntsson, my cofounder of Mullvad, for always helping me find flaws in my reasoning. Thank you to Jonas Magazinius whose feedback motivated me to rewrite the entire paper. Thank you to David Marby and Richard for discussions on Mullvad’s infrastructure requirements, and to Jan Jonsson for many valuable perspectives.

References

This section is not yet complete.
https://trmm.net/LinuxBoot_34c3#Resiliency