Expanding Malware Defense by Securing Software Installations

Weiqing Sun, R. Sekar
Stony Brook University

Zhenkai Liang
National University of Singapore

V. N. Venkatakrishnan
University of Illinois at Chicago

Motivation

- Software installation: more attractive entry point for malware than remote exploits
  - Provides highest privileges needed to
    - Plant rootkits/trojans
    - Hide deep in the system
  - Contemporary OSes don't restrict any actions performed during installation
  - Existing techniques for untrusted code security have largely ignored the installation phase

Assumptions and Goals

- Basic assumption: Mechanisms available for differentiating benign and untrusted software
  - Untrusted software: from untrusted sources, may be malicious
  - Benign software: from well known sources, non-malicious
- Goal: Enable end-to-end life-time defenses against untrusted software
  - Develop policies and enforcement techniques at install/uninstall phases
  - Incorporate existing confinement solutions at execution phase

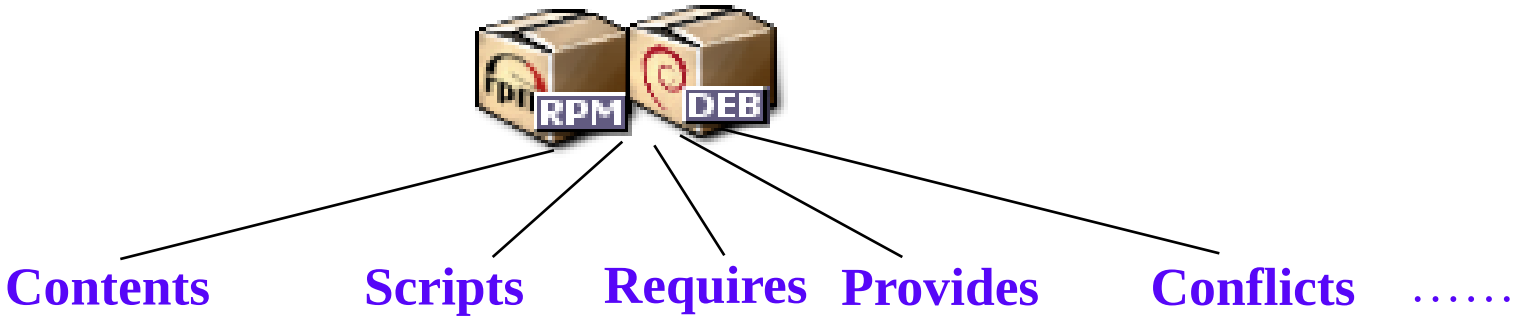
Secure Installation: Requirements

- Security: Untrusted software
  - should not interfere with benign software
  - must always run inside a user-specified sandbox
  - should be *securely* uninstallable at any time
- Usability
  - Installation or operation of benign software should not be restricted in any way
  - Almost all (non-malicious) untrusted software should install successfully
  - Diverse installation mechanisms to be supported
    - Software package managers (rpm, dpkg, …)
    - Self-installing executables
    - Tarballs

Threat Model

- Threats in three phases:
  - Installation phase
  - Execution phase
    - Solutions already exist, e.g., sandboxing
    - *Our goal is to ensure that untrusted code is always run within an administrator-specified sandbox*
  - Uninstallation phase
- Higher-level goal of malware
  - Exploit higher level of privilege during install/uninstall phases
  - Execute code outside of sandbox

Install-time Threats

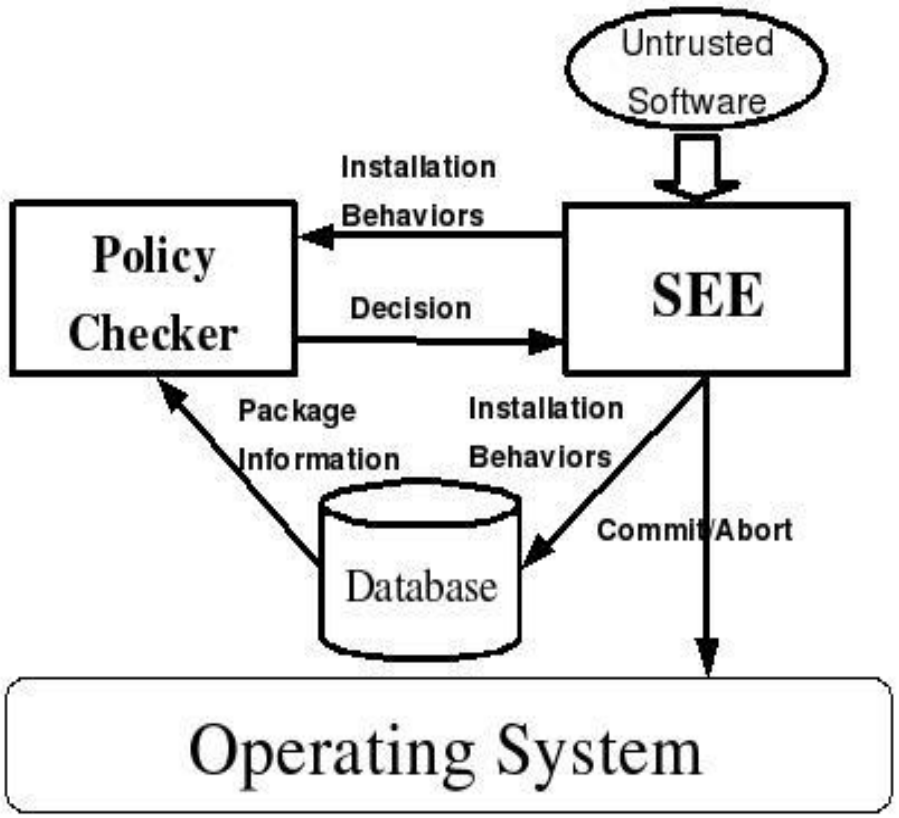


- Attack avenues
  - Perform malicious actions by running scripts
  - Modify files used by benign packages
    - Existing benign packages
    - Benign packages installed in the future
  - Embed attacks in its own files
  - Compromise the integrity of package database

Uninstall-time Threats

- Perform malicious actions
- Compromise package database integrity
- Remove files belonging to other packages
- Leave behind files after uninstallation
- Cause errors during uninstall

Approach Overview



- Initial installation in a virtual environment
- Policy checking
- Commit/abort
- Secure execution phase
- Secure uninstallation phase

Initial Installation Phase

- Need to verify integrity of updates made to critical data, e.g., RPM database
- Two basic alternatives
  - Access control policies: eager enforcement, not easy to determine safety of each update
  - *Alcatraz*, a safe execution environment
    - Installation in an isolated environment identical to host OS state
    - Permits system to go through unsafe states, as long as the end state is "safe"
      - State-based policies are strictly more powerful as compared to "enforceable policies"
    - Supports commit/abort of results observed within Alcatraz
      - Note: Rerunning installation after policy check is unsafe
    - Supports diverse installation mechanisms

Commit/Abort Phase

- Policy verification success → commit
- Policy verification failure → abort
- Commit/Abort functionality is provided by Alcatraz
  - Changes are made to make sure untrusted software run inside a user-specified sandbox

Secure Execution Phase

- Works with diverse confinement mechanisms
  - Policy-based access control
  - Isolated execution
    - All untrusted files (and execution results) stay within a Secure file container (SFC)
  - Dynamic information-flow
    - Label the files belonging to untrusted packages, prevent information flow from them into integrity-critical files
- SSI creates wrappers for untrusted executables/libraries to ensure the use of confinement mechanisms

Policy Checking Phase

- Provide higher level policy primitives to ease development of application-independent policies
  - Can reference package contents and dependencies
  - State (*and* history) based policies
    - Allow modification of ***F*** into ***F'*** such that their diff matches a specified regular expression
  - Action attribution to provide safe exceptions to policies
    - Easier to say that ldconfig is safe rather than to define permissible changes to ld.so.cache
    - Rationale similar to that of DTE, but our implementation leverages Alcatraz to achieve the same effect without OS-support for type enforcement
- Result: One policy for most untrusted software, plus another policy for benign software

Untrusted Package Installation Policy

- No unsafe non-file operations
  - Based on Alcatraz policies with a few exceptions
- No changes to files belonging to benign apps
  - Untrusted installation can only modify/delete files belonging to untrusted packages
- Protect the integrity of package database
  - Modifications must be consistent with the files actually copied to the system
  - Should not change database entries corresponding to other packages
- Grant exceptions based on attribution
  - ldconfig, …

Benign Packages Installation Policy

- Benign packages should not depend on untrusted packages
- No policy enforced during uninstallation time

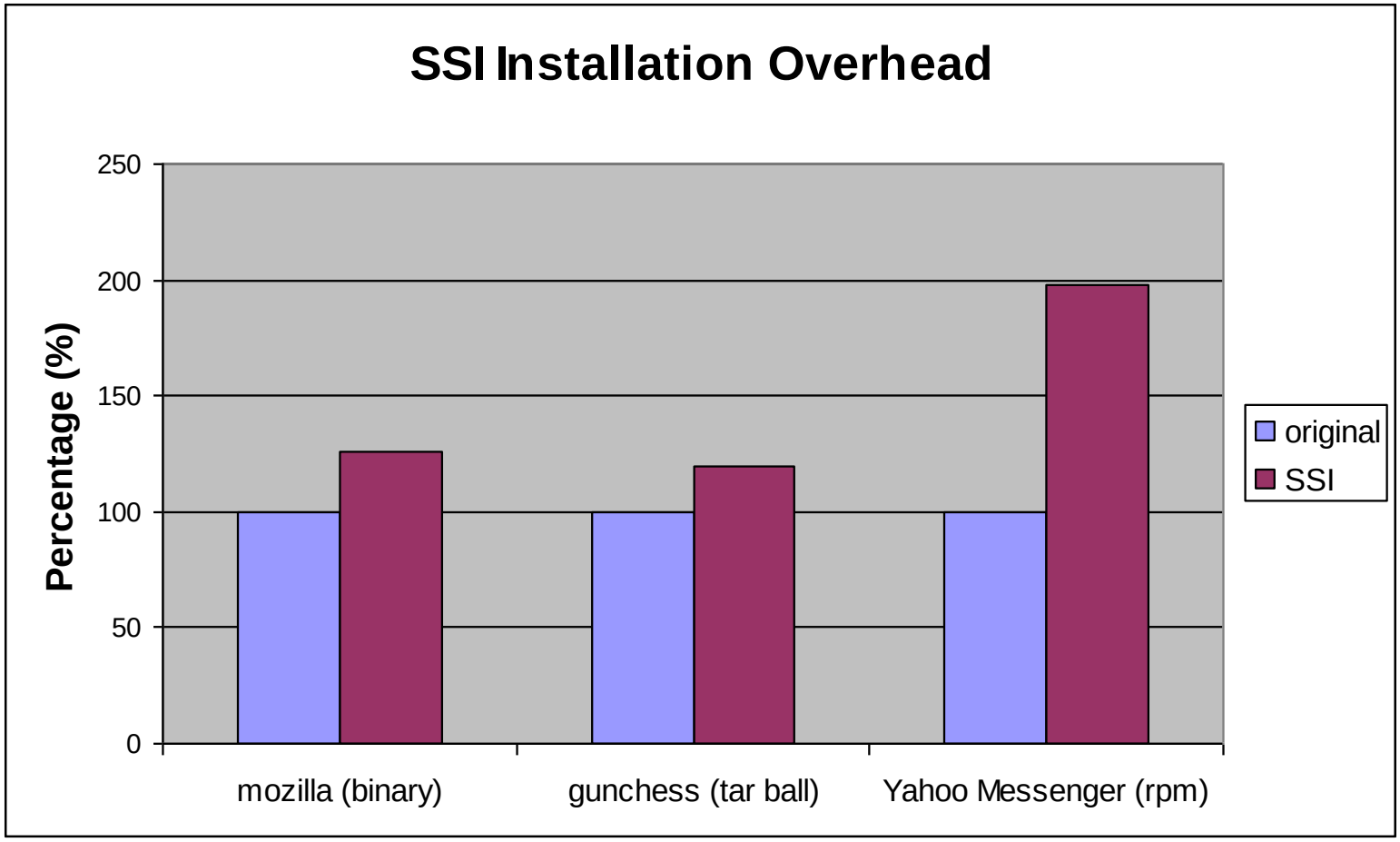
Evaluation

- We have implemented SSI
  - On Linux CentOS 4.1
  - Based on Alcatraz tool
- We have performed installations in SSI
  - Malicious packages (~10)
    - Real-world/crafted, blocked the installations on policy violations
  - Non-malicious untrusted packages (~100)
    - freshrpms/ATrpms, successfully installed
  - Benign packages (~40)
    - CentOs repository, successfully installed

Defeating Malware Using SSI

- Real-world Rootkits
  - Bobkit, tuxkit, lrk5, portacelo
    - Modified files belonging to benign packages (ls, du,…)
- Fake patch from Redhat
  - Created a privileged user with no passwd
- "Malicious" rpm package
  - Crafted rpm package which overwrote glibc and gcc

Performance Evaluation



Related Work

- Software Installation approaches
  - Checkinstall [Eduardo+04]: not for security
  - RPMShield [Venkat+02]: not general
  - SoftwarePot [Kato+02]: not compatible with existing installation methods
- DTE [Boebert+85], SELinux, and Sandboxing
  - Appropriate for confining untrusted software during runtime
  - Not very convenient during installation
    - Every operation needs to be safe
    - Difficulty in policy development
- Information flow based approaches to preserve integrity
  - PPI [Sun+08], UMIP [Li+07], SLIM [Safford+05]
    - Complement with SSI
  - Back to the future [Hsu+06]
    - Recovery needed, availability affected

Conclusion

- Software installation is an attractive vector for malware attacks
- SSI addresses this problem by securing installation process
  - Work seamlessly with execution confinement techniques to “remove gaps in armor”
  - Support a diversity of installation mechanisms
  - Develop high-level policy framework to reduce manpower needed for app-specific policies
  - Evaluation shows our approach is effective and practical

Questions?

Realizing Safe Installations

- Built over Safe Execution Environment (SEE)
  - logically isolates outputs of SEE processes from others
- At the end of installation, check conformance to *state-based policies* to ensure safety
  - Package database modifications to be consistent with state changes observed during installation
  - File accesses to be consistent with trust level of package
    - Untrusted packages can't interfere with benign apps
      - Trojans/rootkits prevented from automatically starting up
  - Prevent some rootkit-like actions
    - e.g., attempt to impersonate a trusted program
- Abort installation if policy violated