Security of Distribution Mechanisms for Linux and BSD Operating Systems

Gabriel Ewing
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, Ohio

Kevin Nash
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, Ohio

***Abstract*—The abstract goes here.**

I. Introduction

An operating system is a software that manages computer hardware resources and provides a variety of services for computer programs. The central core of an operating system, its kernel, is the first layer above hardware itself. Due to of the depth of their functionality, operating systems can potentially yield a great deal more power to an attacker, if the system is compromised, than application software can. Operating systems can be compromised by malicious computer software, such as rootkits, following normal operation, although attacks can sometimes be launched more easily against the distribution process itself, in such a way that users unknowingly install a modified version of the expected operating system. If uncaught, attacks that result in the distribution of a compromised operating system can be both exceedingly difficult to detect and exceedingly powerful in their effect. The open-source model possesses some unique vulerabilities compared to its closed- or shared-source counterparts. Open-source operating systems often have much smaller core development teams than do the popular Windows and OS X systems. Open-source software is not always available on physical media, unlike for example, Windows, which is commonly installed from disc. The storefront websites that are requisite for online distribution of proprietary software also provide a significant barrier that attackers targeting free software would not have to overcome.

...Add more here...

II. Distributing an Operating System

Body of text goes here...

A. Building a Release

The specific processes by which a new version of an operating system is built differ between development teams. Operating system developers establish their own schedule standardizations, though the time between starting the release process and the anticipated release typically takes no more than 90 days. Teams may implement some steps less formally than others do, though the general procedures are ubiquitous enough they can be summarized by a few key steps.

1) Changing the System: In an operating system's natural lifecycle, users tend to discover that the system exhibits certain unexpected behaviors. These bugs are reported, tracked, and hopefully fixed in code so that users will not experience them in future versions of the operating system. Users might request that entirely new features be added or that existing, intentional behaviors be modified. Even without user input, changes to the system inevitably originate from within the development team; design paradigms change and the world of security is ever-evolving.

No matter the cause, an operating system's codebase can change frequently. After enough changes are made to warrant a new release, or at certain milestones in a predetermined development schedule, developers stop making new changes in preparation for a code review. The most developers of different operating systems will "freeze" or "lock" the code so that no more changes can be made. Before the code freeze occurs, developers finish integrating their changes and merge the development branch into a stable, or production, branch. Of course, the naming of branches varies between projects. Once the code is frozen, the interested public and, in almost all cases, an internal body begin a code review.

2) Code Review: At this time, new features are tested for proper functionality, and bug fixes are tested for robustness. In general, the code is reviewed for proper practice and adherence to in-house standards. A beta image is built and distributed to testers. As necessary minor changes are made, such as those to update documentation, fix remaining bugs, and address any discovered security issues, additional beta images will be released for testing.

3) Compiling into ISO: Body of text goes here...

4) Overseeing Release: Body of text goes here...

B. Mirroring a Release

Body of text goes here...

1) Mirror Qualifications: Body of text goes here...

2) Fetching Release: Rsync, Zsync...

C. Distribution Methods

Body of text goes here...

1) HTTP: Body of text goes here...

2) FTP: Body of text goes here...

3) BitTorrent: Body of text goes here...
4) Physical Media: Body of text goes here...

III. Attacks on Distribution

Body of text goes here...

A. "Attack One"

Body of text goes here...
1) Notable Usage: Body of text goes here...
2) Countermeasures: N operating systems currently implement these countermeasures, including Foo, Bar, Baz...
Visual aid goes here

B. "Attack Two"

Body of text goes here...
1) Notable Usage: Body of text goes here...
2) Countermeasures: N operating systems currently implement these countermeasures, including Foo, Bar, Baz...
Visual aid goes here

IV. External Risks

Body of text goes here...

A. Re-Hosting and Ownership Hijacking

Body of text goes here...
1) SourceForge: Body of text goes here...

V. Best Consumer Practices

A. Choosing a Protocol

Body of text goes here...

B. Verifying Mirrors

Body of text goes here...

C. Building a Web of Trust

Body of text goes here...

D. "Soft" Risk Mitigation

Sometimes it is best to rely on proven-stable releases. It can be harmful to be on the bleeding edge of development, although it is a service to the industry.

VI. Our Implementations

Body of text goes here...

VII. Conclusion

The conclusion goes here.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.