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 piece of 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, compromised operating systems can potentially yield a great deal more power to an attacker than application software.

Operating systems can be compromised by malicious computer software, such as rootkits, in the course of normal operation. However, attacks can sometimes be launched more easily against the distribution process itself. This can be done in such a way that users unknowingly install a modified version of the expected operating system. If successful, attacks that result in the distribution of a compromised operating system can be both difficult to detect and powerful.

The open-source software model possesses a somewhat different attack surface that its closed- or shared-source counterparts. Open-source operating systems often have much smaller core development teams than popular commercial systems such as Windows and OS X. Open-source operating systems are rarely distributed using physical media, which is the primary distribution mechanism for Windows. The commercial interfaces that are requisite for online distribution of proprietary software also have advantages and disadvantages in security that are different from the security considerations that open-source distributors must account for.

...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

C. *Hash Collisions*

Providing a signed checksum is part of the best practices for securing download mechanisms. It allows the user to verify the authenticity of the checksum. Once the user is sure that the checksum is the one that they intended to download, they can use the same checksum algorithm to generate their own hash of the downloaded image. If the downloaded checksum and the generated checksum are identical, the user can then proceed, confident that the image they downloaded is the same one that the distribution maintainers published. However, certain commonly-used checksum algorithms are no longer strong enough to reliably verify the correctness of an image.

Weak hashing algorithms introduce a vulnerability to the distribution process. Usually, an attacker that wishes to replace a published image also has to replace the associated published hash with the hash of the malicious image. Signed hashes alert users to this problem. If the hash of the malicious image is the same as the hash of the original image, though, the attacker does not need to replace the published hash and signing mechanisms are no longer sufficient.

Hash collision attacks are particularly dangerous because normal protection policies do not detect their presence, allowing a malicious image to be installed silently. There are ways to detect a successful hash collision attack: one is that the malicious image would be different upon comparison to the original published image. Another is that distributed source code can be independently built and compared to the malicious image. If the distributed source code has also been replaced, then a code review or version control logs could offer clues. However, none of these checks is part of the standard practice recommended by an distribution that we surveyed. Because of this issue, weak checksum algorithms are a very dangerous practice.

The MD5 hash algorithm was published in 1992 by Ronald Rivest [1]. Weaknesses in the algorithm were quickly discovered [2] and MD5 quickly dropped out of favor among the cryptographic hashing community. In 2004, a collision of the full MD5 algorithm was published [3]. MD5 collisions were used in proof-of-concept attacks on certificate authorities as early as 2005 [4]. While finding a collision that successfully installs and executes useful malicious tasks is more difficult than simply finding a collision, computing power has increased considerably since 2004 and it is not unreasonable to suspect that this capability exists in the world today.

Unfortunately, many of the of the distributions that we surveyed, including Mint, Ubuntu, Debian, Arch, Slackware, and Mageia offer MD5 checksums. While none of them offer only MD5, many list it as the first option. MD5 is faster than the SHA variants, and is reliable in detecting random corruption from transmission, but the time to compute a checksum using a strong hash algorithm is still negligible compared to the time to download the image on any modern hardware. We consider offering MD5 checksums a poor practice because users with minimal cryptographic background may choose to only verify using MD5 and trust that a hashing algorithm used by the distribution is secure. Replacing an image and MD5 checksum would remain undetected until a stronger algorithm was tried and the distribution maintainers were notified.

1) Countermeasures: Distribution maintainers should remove MD5 checksums from their download pages and offer checksums generated by state-of-the-art algorithms. Additionally, they should compare the images offered by mirrors and their main download server to the image that they originally built on a regular basis. Users should verify downloaded images using the strongest algorithms available.

IV. External Risks

Body of text goes here...

A. *Re-Hosting and Ownership Hijacking*

Body of text goes here...

1) SourceForge: SourceForge is a hosting service for open-source software. In 2015, the website was accused of bundling malware with the binary project packages that it offered for users to download. This caused some large projects to abandon the site entirely [5]. While the company later announced a change to this policy [6], accusations have continued [7] and SourceForge can no longer be considered a trustworthy hosting platform.

Manjaro Linux was the sixth most-popular Linux distribution between March 2015 and March 2016 according to DistroWatch [8]. According to our research, SourceForge is the only download platform recommended and offered by Manjaro. While downloads are available via torrent, the torrent links are hosted by SourceForge as well and so the original seed may be of a compromised image [9]. We were unable to locate any alternative download mechanism.

This means that there is a real possibility that as of this writing, image downloads of Manjaro are bundled with malware. We recommend that users avoid downloading, installing or running Manjaro Linux until the distribution maintainers review their practices and provide alternatives for downloading.

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] R. Rivest, "The md5 message-digest algorithm," 1992.

[2] B. den Boer and A. Bosselaers, "Collisions for the compression function of md5," in *Advances in CryptologyEUROCRYPT93*. Springer, 1993, pp. 293–304.

[3] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions md4, md5, haval-128 and ripemd." *IACR Cryptology ePrint Archive*, vol. 2004, p. 199, 2004.

[4] A. K. Lenstra, X. Wang, and B. de Weger, "Colliding x. 509 certificates," Tech. Rep., 2005.

[5] M. Schumacher, "Gimp project's official statement on sourceforge's actions," May 2015, accessed April 2016. [Online]. Available: https://mail.gnome.org/archives/gimp-developer-list/2015-May/msg00144.html

[6] "Third party offers will be presented with opt-in projects only," accessed April 2016. [Online]. Available: https://sourceforge.net/blog/third-party-offers-will-be-presented-with-opt-in-projects-only

[7] "Sourceforge hijacks the nmap sourceforge account," 2015, accessed April 2016. [Online]. Available: http://seclists.org/nmap-dev/2015/q2/194

[8] "Distrowatch page hit ranking," 2016, accessed April 2016. [Online]. Available: https://distrowatch.com/dwres.php?resource=popularity

[9] "Download manjaro," accessed April 2016. [Online]. Available: https://wiki.manjaro.org/index.php?title=Download_Manjaro