SECURITY HARDENING OF A LINUX-BASED PRODUCT

Aditya Lad
Senior Software Engineer
Anand Subramanian
Principal Software Quality Engineer
EMC Corporation

Table of Contents

Abstract 3
Introduction 3
Gather Your Resources 3
Threat Modeling 4
Prioritizing Product Areas for Security 6
  Application security 6
  Component security 7
  OS-level security 8
  Security at Network Level 9
  Cryptography and Security of Network Communication 10
Help from Vulnerability scanners 11
Security Testing 12
  Getting Help from Automation 12
  Help from Penetration Testing 13
  Tracking and resolution of security bugs 13
Importance of security training 13
Summary 14

Abstract

Securing Linux is not an easy job. Even though securing software is a never ending task and in fact true security does not exist, it is an important requirement of today's software development. Good security practices never go unnoticed. This article discusses the various points which must be considered while strengthening the security of a Linux-based product. The article's primary focus is to suggest means on how to approach your product's security and execute it. The attention on the technical details and nitty-gritty is minimal, as a lot of good sources are already available online. The content is based on practical experiences accrued over a period of time.

Introduction

Setting out to harden a Linux-based product can be an intimidating task, considering the myriad dimensions to security. This article jots down the list of high level points a practitioner needs to adopt for successful hardening. The ideas have been accumulated as a part of securing Linux-based products over a period of time. Many of these thoughts are not specific to the Linux operating system as such, and can be applied to any Operating System; technical implementation might vary, however.

The article starts with the planning for hardening of your product, followed by the technical areas that generally require attention during implementation. In the end, it discusses the testing and other approaches with an intention to verify your hardening efforts.

Gather Your Resources

Section Highlights

- List the scope of hardening based on your business needs
- Research available sources of information
- Take input from stakeholders and security experts
- Be objective about what is expected
- Track planned vs. accomplished via Security Audits

Since timely release of the product often means aggressive deadlines, it is important to not get lost in the details especially since security hardening is more of a research-oriented subject. It's very important to have the right resources and the scope and priority of hardening identified and agreed upon before starting the implementation. Much of this depends on the nature of

deployment, the business use case of your product, and stakeholder expectations. For example, Internet-facing systems are at far greater risk than systems deployed inside an internal network. If your product does not use web applications, then there is no point researching web application attacks and mitigations. If your customers are concerned about unencrypted communication in your product, it might make sense to prioritize a plan to address network encryption.

Input can be gathered from product security team guidelines and policies in your organization, vendor-provided security guides, general security books, and discussion with the project stakeholders. Remember, choosing good, objective standards and references will always help you justify and explain your security decisions later.

At the end of the development lifecycle, the product should be submitted through a security risk assessment, preferably by an independent group of experts. This audit would certify the product for a secure release or reveal areas that are critical to product security and need to be fixed before releasing to customers.

Threat Modeling

Section Highlights

- Create Threat Models for your product
- Component-level vs. System-level Threat Model
- One time effort, long term rewards

Prepare threat models for the business workflows involved in your product. Threat modeling is an extremely important and effective means of evaluating the security of your product. Threat models can be component- or workflow-specific that can capture a lot of implementation details and could, at system level, capture component details and major technologies involved. Preparing a detailed data flow diagram for your system or component would be a good way to start building a threat model. Figure 1 is an illustration of the data flow diagram for a product.

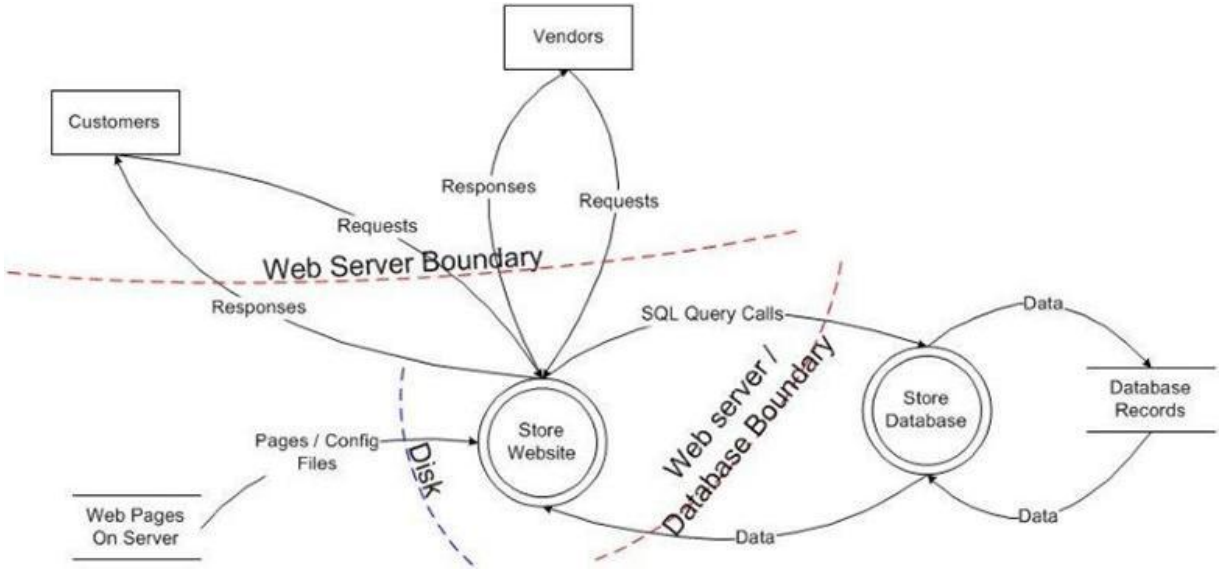

Figure 1: Sample Data Flow Diagram[1]

Threat models are prepared at system level for the entire product to capture interdependencies between components in the system. Threat models can be prepared at the component-level for products comprised of multiple components, have distributed client-server architecture, or dependencies amongst modules. For instance, a component-level threat model can include an application server and its communication with the database. It could identify threats associated with the use of an application server, Java application, and its interaction with the database. It could also contain implementation-specific details such as the use of JDBC, input validation in the applications, security against database-oriented attacks such as SQL injection, and so on, whereas a system-wide model could capture the threats at the system level. For example, the mode of communication of a server with the clients, the version of SSL being used, cluster communication, possibility of man-in-the-middle attacks, physical security aspects, and so forth.

A system-wide threat model requires a complete architectural understanding as well as the business use cases of the product, whereas the component-level model could be created by the responsible developer and tester. Although preparing threat models is a one-time detailed effort, for a secure future, it is worth the effort. Threat modeling training should be available as a part of product security training. If you plan to do the threat modeling exercise for the first time, don't do it without the help and guidance of a security expert.

[1] Source: http://resources.infosecinstitute.com/application-security-deconstructed

Prioritizing Product Areas for Security

Section Highlights

- Classify areas that need security focus
- Prioritize, plan, and have a mitigation strategy

Identify the most risky areas of your product. Typically, these areas directly handle network communication or process user inputs. Prioritize them based on the risk level and complexity involved. Analyze the efforts each area would require to research, learn, and run security tests. Then, based on the time available, plan your efforts in a timely manner. Identify the associated risks and look for mitigation plans. For example, if you decide to harden the database, is there any chance that one of your security measures might actually break some other functionality or might bring down product performance?

Mitigation plans might include regression and performance tests or, in worst case, removal of such a stringent measure with mitigation plans at some other layer. The next few topics might help you identify such areas. Broadly these are categorized into:

- Application Security: deals mostly with code level problems in a web application-based product.
- Component Security: deals with individual third party products such as web servers, application servers, and database.
- OS-level Security: deals with the best security practices inside an OS.
- Network-level Security: deals with the way the product as a whole communicates over the network.
- Cryptography: deals with the good use of encryption in the product.

Application security

Section Highlights

- Be cognizant of prevalent web application vulnerabilities and attacks
- Input validation is the key
- Be aware of security configurations in underlying software
- Follow industry best practices such as those recommended by OWASP

Application security is a mammoth task requiring an altogether different area of expertise. The good old days of exploiting telnet, FTP, and sendmail were over a long time ago. These days, web application vulnerabilities are the major entry points for attackers. A majority of

vulnerabilities involve insecure coding practices, input validation, and output sanitization issues that could be avoided with the right security training, source code security analysis, and security awareness of common attack vectors.

Others are attributed to security (mis)configurations while using the underlying software. The ability to find a vulnerable server using powerful search engines makes an attacker's job even easier. When security vulnerability is exposed in a CMS, it is typical for an opportunist attacker to locate such servers all over the Internet running the affected version of the CMS and try his/her exploits. Although it is impossible to stay ahead of unknown threats, following the best web application security practices always mitigates the risks to a reasonable level and makes an attacker's job difficult.

The OWASP top 10 project list includes the prominent 10 mistakes that pose web application security risks.

Component security

Section Highlights

- Choice of right software version based on your business needs
- Disable unwanted features and modules that will not be used
- Document and maintain component-specific security measures adopted for future reference

Component-level security refers to the major software components that are installed in the system for web servers, database, application servers, and so on. Each of the components should be carefully analyzed for security best practices. This starts with choosing the correct version of the software to suit one's security requirements; generally, the latest available version. The options used and features enabled during the compilation and installation phase should be carefully selected. Do not install any component or module which is not required. For example, the default Apache web server installation comes with a number of modules and features which a typical application does not use at all. The removal of unnecessary modules improves software performance by reducing the memory footprint and also limits the attack window. A similar rule could be applied to application servers, databases, and so forth.

Since every software has its own best practices, security risk levels, and complex security mechanisms, it is always good to maintain a list of security measures separately for each component. Many times it is not possible to take certain security measures due to business or

technical complexities. Don't be too casual about such occurrences. A security-savvy customer might haunt you with escalations in future. In such cases, it is useful to have a pre-emptive mitigation plan and strong justified reasoning. It is best to document such things and the mitigations after a discussion with the stakeholders as known issues, so that you are not caught off guard. Over time, such a list will provide great value in understanding the security needs of your business.

OS-level security

Section Highlights

- Trim OS to a bare minimum and retain only necessary RPM packages
- Review system and network services, disable unwanted ones
- Remove unnecessary startup scripts as needed
- Be extra careful with critical system accounts, such as root
- Ensure kernel versions are kept up-to-date
- Exercise care when enabling ssh, configuration parameters, and environment variables
- Ensure you have enabled the right level of logging for critical components
- Refer to industry best practices and checklists, such as DISA STIGs (http://iase.disa.mil/stigs).

Securing an operating system such as Linux could be a daunting task. The OS includes the kernel, system libraries and packages, necessary partitions, and application packages. To start, minimize the number of packages installed in the system. Remove what is not required, keeping only the bare minimum required to run a system. Especially, keep checks on the network-related services running on your system. Disable—or, ideally, remove—services that are not required. If you cannot remove them, look for the vulnerabilities associated with their versions and secure their configuration or patch them to their latest versions. Ensure that the startup scripts are thoroughly cleaned and no unknown service is enabled to start by default. Ensure the kernel version is up to date and does not have any serious security problems.

However, the most important part while securing a Linux internally comes with the customizations and configuration. Care must be taken while using utilities such as ssh keys and features like chroot, sudo, and so on. File permissions, configurations, and available environment variables should be carefully set. Many of the protections can be easily defeated if these are not configured properly. Do not keep developer tools such as gcc compilers, perl or python packages, and network-oriented tools like FTP, telnet, wget, tcpdump, and so forth if

they are not required. These can make the job of an attacker easier. Another good reason to remove them is if your product is aiming for a PCI sort of compliance. Keep the OS updated with required security patches and also thoroughly test the features through regression after a patch update. Keep a knowledge on the cryptographic modules being used in the system for the generation of entropy. Ensure you are following the best practices. Harden your system through strong password policies and effective lockout policies. Ensure the superuser accounts (e.g. root) are not set to use default or frequently used passwords nor that such critical accounts are accessible through the network.

Do not underestimate the power of logging. Some important system related events which might help you immensely could be system events, kernel messages, boot messages, ssh events, privilege escalations using su and sudo use, last logins and reboots, cron events, and so on. Be sure to have the required level of logging, right set of permissions for logs, and a rotation policy to ensure that logs do not fill up the disk space.

For additional help and guidance, get the DISA STIG (Defense Information Systems Agency Security Technical Implementation Guides) for RedHat Linux. It contains a checklist of around 550 items, which can be checked and tested by an average Linux admin. The exhaustive list covers configuration issues, best practices, and recommended settings for traditional Linux components such as Apache web server, Openssh, and so on. If even your product does not run on RedHat, it is easy to find out the corresponding setting for other flavors of Linux as well.

Security at Network Level

Section Highlights

- Be wary – this a favored entry point for any attacker
- Double-check firewall rules
- Use port scanners to determine system information with and without firewall
- Shut down open ports or document their use

The first step for an attacker is to find information about your system through the network. The more careless you are about your network security, the more likely it is for an attacker to find a vulnerable service that could help him/her get inside the system. Removal of unnecessary services is already covered in the OS level security.

Configure the firewall to allow traffic only on the required ports. Inspect the firewall rules carefully. Make sure you do not ‘Accept’ some protocol that is not even required. Even you do

not have IPv6 enabled in your system, do not keep a firewall rule to accept IPv6 traffic. If your product requires a few ports to be opened, try to open them only for registered or trusted IP addresses in the system. Use a port scanner with various options to run different flavors of port scans to aggressively check how your system looks from the outside when the firewall is running and when it is turned off. You can also use the port scanner to see how much information your system 'leaks', e.g. running configuration and versions of network services. Use OS tools to find what ports are being used by which processes and document them.

Cryptography and Security of Network Communication

Section Highlights

- Know-how of cryptographic services running on the system and their configuration
- Use of strong encryption algorithms and key sizes during encryption and certificate generation
- Use SSL scanning tools to verify the nature of encryption being used

How do the crypto modules in your Linux handle the randomness? How good is the entropy pool in your Linux? What sort of encryption is used by the network services in your product? How good is the key strength? What kinds of ciphers are being used by your SSL services? Does it support the latest version of TLS? Why are you using md5 certificates? These are some of the questions a security-savvy customer might ask you point blank. And chances are, we don't have answers to many.

Encryption and decryption has been there from the days of Julius Caesar. Today, cryptography in the OS has evolved to a far greater scale and is an important feature on which multiple components depend and should not be ignored simply as being too complex. Entropy pool in Linux is considered as the amount of randomness present in the system. In simple terms, randomness comes from a series of complex algorithms and procedures that translate keyboard events, network activity, and other 'non predictable' events into a random sequence of bits. Cryptographic services use this random pool to generate random strings. Obviously, the greater a system's amount of randomness, the less likely an attacker can predict it using known cryptographic attacks. Even though the details are quite complex to understand for an average computer user, it is good to have a general idea about this mechanism in Linux, the importance and the use of devices such as /dev/random, /dev/urandom, and random number generators that keep entropy levels of a system high. This is especially true if your product deals with security and, even more importantly, if your product deals with cryptography.

For encryption security, it is advisable to use strong and recommended encryption algorithms and key sizes during encryption, certificate generation, and so on. If you are using SSL or TLS, ensure that you support the recommended version. Older versions are known to have protocol-level weaknesses. For a general understanding of certificates and popular algorithms, we recommend training and books related to Public Key Infrastructure (PKI) and general cryptography. Over time and due to advances in this field, sometimes algorithms give way to newer and stronger algorithms. A general knowledge on strong and popular algorithms is helpful.

In addition to this, use SSL scanners to verify that the SSL services offer quality encryption algorithms and key sizes to the clients and are not vulnerable to severe attacks.

Help from Vulnerability scanners

Section Highlights

- Use vulnerability and source code scanners regularly
- Analyze the reports and fix the findings
- Get the false positives documented

Using a vulnerability scanner to regularly scan your system is always a recommended practice for securing your system. A vulnerability scanner is an expert program which checks your system for the latest vulnerabilities. It gives you a good idea on the amount of information leaked by your system to the outside world. A scan can easily check for the common 'not-so-good' configurations, vulnerable versions, and can provide a good starting point for securing the system. The reported problems should be prioritized as per their severity and fixed accordingly. The choice of scanners, the way the scan is performed, and the analysis depends on the product and the way it is expected to be deployed in a production environment. Commonly known scanners such as Nessus can perform a network scan on the OS, an internal scan, or even a DISA STIG scan. DISA STIG scan is applicable for products that have qualified for it; e.g. Redhat and Oracle. Qualys scanner performs scans similarly whereas Cenzic scans can scan the web application URL of your product for known web-based vulnerabilities.

Source code scanners can review your source code for known insecure coding practices. An example in this category is the Fortify scanner. Run the scanners at regular intervals during the product development phase and analyze the reports for the reported items. Most importantly, ensure that the scanners are updated with the latest vulnerabilities and plugins available at the

time of run. Do not rush to fix the reported items without a careful analysis of its root cause and impact on the product as a whole. Often scanners report items based on the software versions they see in the product, while it is quite possible that you may not be using the affected module itself. In such cases, you should document the behavior as a false positive with a sound technical justification.

Security Testing

Section Highlights

- Create a plan for security testing
- Keep updated with the latest attack techniques and mitigations
- Subscribe to famous security lists and security news websites

Have a plan for the security testing of your product. Security testing is very different from traditional testing. It involves far more research and understanding of the underlying technology along with a passion, persistent will, and curiosity about the security technology, often sometimes referred to as the 'hacker mindset'. At the same time, the tester should be aware of the prevalent attack vectors in the security world for a given technology. Almost daily, new techniques evolve to defeat the security of a system. Security has evolved rapidly in the last decade thanks to security awareness of the computer industry, and it is quite possible that a majority of the old security test cases that you execute could be obsolete and considered impractical in today's environment. Subscribing to well-known security mailing lists and websites would certainly help in this regard and will keep your knowledge current.

Getting Help from Automation

Section Highlights

- Save time by automating regression test suites
- Automate dumb security testing

Although an ideal way of performing security testing is to carry it out manually by an experienced security tester. However, for practical reasons, you can always use automation for low priority issues, issues that have been fixed in past releases, and issues that are time consuming. A successful run of a good automation suite would boost confidence in your product and also enable early detection of security misconfigurations produced during the development phase of the product.

Help from Penetration Testing

Section Highlights

- If possible, get penetration testing done on the product

Penetration testing is a form of security testing where an experienced professional hacker or security expert tries to break the security of your product. Such testing, when performed by skilled professional, increases the confidence in your product. Needless to say, this should require a high degree of trust and contractual binding with the tester so that he/she does not misuse any information. In order to make the most out of the testing effort, one should plan it when the majority of product development has been done and security hardening and existing security bug fixing has been performed.

Tracking and resolution of security bugs

Section Highlights

- Track security bugs diligently during product development
- Ignored bugs can turn into vulnerabilities tomorrow

During product development, security-related bugs should be tracked with due diligence. The bugs could be reported during security scans, source code scans, regular security testing, third party penetration testing, and so on and reported by customers in previous releases. There is a separate topic—called vulnerability response—which is more specific to the vulnerabilities found in systems in production use. In contrast to such vulnerabilities, the bugs we refer to in this section are those found during the active development of the product. You might have already guessed it right; any security bug ignored or missed during the development phase is a potential vulnerability and will certainly create a lot of problems for your customers.

Importance of security training

Section Highlights

- Attend security training regularly
- Choose training based on the technical area you work on
- Make sure to have a hands-on experience in exploitation and vulnerability fixing

Computer security is a vast field and perhaps it is impossible for an average user to understand the evolution of security in a short span of time. At the same time, it is important for an average

computer user—especially those in the software development business—to be aware of the best security practices.

Security training in an organization is the most easily available mode of learning. Good training not only makes the user aware of the hazards of insecure programming, but also prepares them to implement what they have learned in everyday use.

In the case of software development, this can only be possible if the developer is able to relate the learning with the kind of code he/she writes. For developers, training that focuses on workshop and hands-on experience are generally considered better and interesting. Typically in such training, the developer should be educated about a certain sort of attack, well known or infamous occurrences of that attack, and the business impact it caused, followed by a hands-on experience to fix the vulnerable code.

Summary

While hardening and security appear overwhelming at the outset, it can be made easier by following a methodical approach with proper planning, prioritization, execution, and validation throughout the product lifecycle, as described in this article. Secure product development can become a way-of-life by creating awareness through training and hands-on experience.