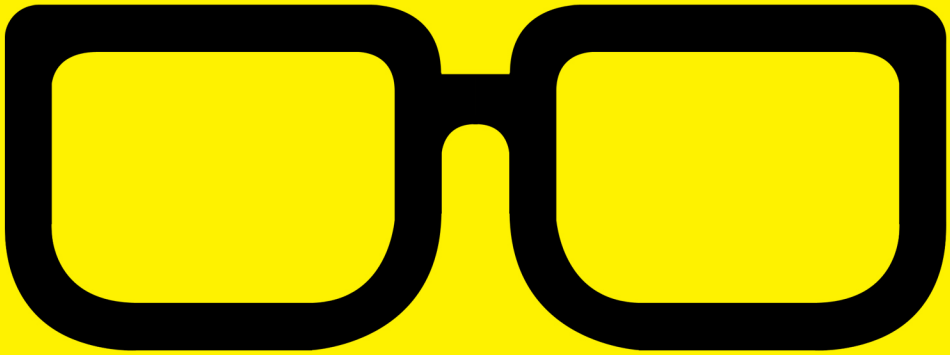
SPONSORED BY

Bit9 + CARBON BLACK

GEEK GUIDE



Linux in the Time of Malware

LINUX™ JOURNAL

Table of Contents

FEDERICO KEREKI is a Uruguayan systems engineer with more than 25 years of experience doing consulting work, developing systems and teaching at universities. He is currently working as an UI Architect at Globant, using a good mixture of development frameworks, programming tools and operating systems—and FLOSS, whenever possible! He has written several articles on security, software development and other subjects for *Linux Journal*, IBM developerWorks, and other Web sites and publications. He wrote the *Essential GWT* book, in which you can find some security concerns for Web applications. You can reach Federico at fkereki@gmail.com.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About Bit9 + Carbon Black

Bit9 + Carbon Black provides the most complete solution against advanced threats that target organizations' endpoints and servers, making it easier to see—and immediately stop—those threats. The company enables organizations to arm their endpoints by combining continuous, real-time visibility into what's happening on every computer; real-time signature-less threat detection; incident response that combines a recorded history with live remediation; and prevention that is proactive and customizable.

More than 1,000 organizations worldwide—from Fortune 100 companies to small enterprises—use Bit9 + Carbon Black to increase security, reduce operational costs and improve compliance. Leading managed security service providers (MSSP) and incident response (IR) companies have made Bit9 + Carbon Black a core component of their detection and response services. Visit http://www.bit9.com for more information.

Linux in the Time of Malware

FEDERICO KEREKI

Introduction

The malware threat landscape is expanding continuously, and in the past 30 years since the appearance of the first virus, the number of security events (and their associated financial losses) has continued to grow at an exponential rate, with millions of such events per month. These numbers support a somewhat fatalistic opinion that says there are only two kinds of systems: those that already have been attacked and those that will be attacked.

The Malware Menace

Malware is software designed to gain access, gather information, engage in illicit operations or disrupt normal operations. The word itself, a portmanteau of "malicious" and "software", points to its goals. Malware takes many forms, although most have an economic objective.

Custom malware directly targeted against a specific system will be far more dangerous than "generic" malware, so defending against that type of attack is critical. Research confirms that more than 70% of all malware is specifically tailored and used only once, so the threat level stands high.

Rising Risks: It's safe to say that many Linux users believe their systems are "secure by design", malware-safe or even that there are no credible threats against their servers.

Getting actual numbers regarding concrete security incidents is difficult, because many incidents are not disclosed or made public, but some general statistics are available that shed light on this growing problem. See, for example, Verizon's Data Breach Investigation Report at http://www.verizonenterprise.com/DBIR/2015.

Some of these beliefs are at least partially true; for example, the numbers suggest that a default Linux system probably *is* more secure than other systems. Additionally, Linux environments have seen few attacks, such as are commonly reported for Windows. In any case, assuming that attacks on Linux servers *never* happen or have no chance of success is not only false, but also dangerous.

Malware authors have, for the most part, targeted

Windows machines, and the main reason is simply on desktops, Windows presents a larger opportunity with the huge number of workstations. But on servers, it's a different story, and we are increasingly seeing new and targeted malware attacks on Linux, such as the 2014 Windigo attack that infected 10,000 Linux systems and the more recent "Mumblehard" malware that was discovered by ESET and operated for at least five years. In both these examples, the malware was targeted, allowing it to operate under the radar and avoid detection.

Thus, when considering the server space, Linux's popularity becomes a big attraction for would-be attackers. Due to the sheer numbers of Web servers and database servers that run under Linux (some statistics say approximately 70% of all servers connected to the Web run Linux), a malware author who succeeded in targeting Apache, MySQL or similar server software immediately would have a "target-rich" environment, in military terms. Given that many business have critical corporate information and systems running on Linux servers, the menace becomes even more serious; such an attack vector could help target a specific company, with a planned stealthy invasion.

Rather than taking a "shotgun" approach with malware and trying to bag as many machines as possible, even if low-valued, attackers usually prefer going sniper-like after bigger, more attractive targets. It's just like "Sutton's Law": reputedly, bank robber Willie Sutton once was asked why he robbed banks, and he simply answered "Because that's where the money is." It is estimated that around two

thirds of all malware attacks are used only once and are specifically directed against given companies or servers. (And, of course, robbing a bank is likely better than mugging 100 random people on the street!) So, even if your servers are reasonably well protected against general, run-of-the-mill, common attacks, it certainly will pay to consider the possibility of a direct, well-tailored, malware attempt against your specific company.

Finally, "zero-day" attacks should cause the most worry. These attacks are based on undetected software vulnerabilities that haven't been patched. There is no awareness of them and no knowledge of their invasion methods, their effects or the best way to eradicate them.

Zero-day threats, despite their name, can be exploited for longer durations if left undetected and unpatched. For example, the Heartbleed bug affecting OpenSSL originated in 2011, but it surfaced almost three years later. As another example, in 2008, Microsoft confirmed a specific vulnerability in Internet Explorer that dated back to 2001. Software for which support has ended with no more available patches also are candidates for zero-day exploits.

Entry Points: In "olden" times, a three-legged system architecture based on the Internet + DMZ + Intranet trio was enough for security purposes. However, the current communications landscape, with VPNs, work-from-home users, Wi-Fi connections, trusted partners, open ports, cloud servers, service-oriented architectures (SOAs), remote management, port hopping and more, certainly is much more complicated and porous.

Passwords and a firewall may have been enough previously, but this variety of possible entry points greatly adds to the difficulty of protecting a server against malware. In fact, you should assume that your network perimeter, instead of being hard and solid, is soft and permeable, and plan accordingly for the worst.

Of course, all of this doesn't mean you should forget about, say, SSH (secure shell) connections or about using tunneling for remote access. Security is based on setting as many obstacles as possible ("defense in depth") in the way of would-be intruders.

Traditional Defenses

Sysadmins are savvy about possible attacks—at least, they are much more so than common users—and this, coupled with the fact that Linux systems tend to be viewed as more secure "from the get-go", means that servers often have some level of protection against nuisance or common malware. Defenses often are set up after the fact, and more advanced defenses are not planned or put in place until a breach has been discovered. Managing defenses depends on many separate programs, processes and tools, and it isn't a simple job. It can take plenty of time, no matter what levels of automation are achieved. And finally, if not properly considered at the beginning, it can be difficult to keep up with new threats and ensure that all security methods are running properly.

Attack Surface: Intuitively, you can define the attack surface of any system as the sum of all the ways (attack

vectors) that malware can invade the system. The larger the surface, the more insecure your system proves to be. Table 1 shows the various aspects of an attack surface.

TABLE 1. Attack Surfaces

ASPECT	RISK
Human	Social-engineering attacks may dupe valid users into infiltrating the system themselves, unwittingly but directly injecting malware into your servers. Even an e-mail message, opened by an unaware user, can become an attack vector.
Hardware	The workstations themselves—although attacks on these usually require some inside help or internal proximity. (Google the TEMPEST codename and learn about its spying methods.)
Network	Open ports and available services and interfaces on outward-facing servers, or code listening on visible ports, provide possible entry points to your system. In particular, technologies based on tunneling or peer-to-peer connections are a threat, because they open direct connections to systems. Depending on your architecture, you may need to add cloud servers and systems to the network attack surface.
Software	All running code may include exploitable vulnerabilities. Even worse, computer systems may secretly and deliberately include “backdoors”, allowing remote access to parties in the know. In this context, Web applications are the most worrisome problem, because they usually depend on several pieces of software (Web servers, database servers, content management systems, e-business packages and so on), each of which may have vulnerabilities.

You also should consider host-based detection and response tools that can provide real-time detection of attackers or malware.

In today's data center, firewalls are usually the main defense against malware. But, recent breaches have shown them to be increasingly ineffective, and they should be supplemented with host-based defenses. For example, in a Linux environment, brute-force attacks against an SSH port can be detected with log analysis tools (DenyHosts or Fail2Ban are a couple possibilities), and higher-level access rules then can be implemented automatically to thwart the would-be intruder. You also should consider host-based detection and response tools that can provide real-time detection of attackers or malware.

Directly reducing all these aspects—adding physical security, having fewer services running or closing down ports—minimizes exposure and lessens the risks of attack, although obviously you can't go too far without enduring severe loss of functionality. Attack surface reduction helps prevent malware entry into your system, but it doesn't help with actual damages in the case of a security failure.

Access Controls: Access controls provide ways to limit access to systems and resources to reduce their attack surface. Table 2 shows the different categories of access control methods.

TABLE 2. Access Control Methods

ACCESS CONTROL TYPE	DEFINITION
Mandatory Access Control (MAC)	Restrictions on a user's ability to access some resource or perform some operation. Users are not allowed to grant rights or override the policy, which is centrally managed by an administrator. SELinux (Security Enhanced Linux) is a Linux kernel security model that supports MAC for more secure Linux systems.
Discretionary Access Control (DAC)	A system that restricts access to features based on the identity of users or the groups to which they belong. The controls are said to be discretionary, in that a subject may pass his or her own permissions to other subjects, unless restricted by a MAC rule. The Linux filesystem, with user and group permissions, is an example of a DAC.
Role-Based Access Control (RBAC)	A more modern technology that grants access to resources depending on roles, which are assigned to users based on their job functions. This is an application of the "least privilege" principle (which I discuss later in this eBook).
Rule-Based Access Control (confusingly called RBAC also)	A way to allow or deny access to resources based on rules defined by an administrator. When a particular user (member of certain groups) wants to access some resource, its rules are checked to see whether it can grant access. A simple example of rule-based access control is permitting access to a database only at certain hours of the day, from Monday to Friday. A potential problem with this method hinges on the number of events and resources that need rules.

Often one or more of these methods are active in a given system, so they provide a first barrier against malware exploits, but you need more defenses that that, as you'll see.

Signature-Based Detection: Whenever suspicious software is detected "out in the wild", it's analyzed by researchers for antivirus companies. After confirming that it's actually malware, a "signature" (a recognizable sequence of bytes) is taken from the code and added to the antiviral database, so future recognition is assured. New or custom malware attacks or zero-day exploits won't be detected by antivirus software, since the required signatures won't yet exist. Plus, according to FireEye research, more than 70% of malware is highly targeted and used only once.

Signature-based approaches and network defenses have some other problems as well. Malware writers can encrypt their code so the actual malware will be decrypted and executed afterward, but it will be harder to recognize. In effect, there can be uncountable versions of the same malware, each one encoded with different keys, making it harder to detect.

Antivirus software is quite common on Windows machines, but Linux users tend to dismiss it out of hand, assuming that kind of malware is not relevant, and this makes sense when you consider that companies often are slow to develop signatures for Linux machines. But, this is a rather shortsighted point of view.

Principles of Modern Defenses

As cyber attacks evolve and attacks on Linux systems become more common, defenses against malware must

evolve from a reactive process to a proactive one. The usual cycle in companies is Prevention→Detection→Solution, but the evolution of modern threats has tempted some people into believing that prevention is nearly impossible and that "detection is the new prevention". This sort of misses the point, because a more profound look at your systems and at modern defense methods can provide a high level of "first-line" stopping power, and the addition of continuous, real-time, monitoring can add extra capabilities to deal with the sheer volume of sophisticated, often targeted malware as well as the diverse and multiple attack vectors (users, mobile devices, the cloud and so on). Let's look at some principles that are mandatory in modern security planning.

Zero Trust: The classic model for network security was like a medieval castle: a great guarded wall (possibly with a moat) that supposedly no one would be able to get through, with an open space inside where life could go on safely in a normal way, even during sieges and wars. However, for a modern computer center, this analogy breaks down, because you can't ensure that nobody will be able to get inside where all the internal company resources will become available for pillaging.

The zero trust principle, developed by Forrester Research, drops the idea of an untrusted external network and a trusted internal network, and instead assumes that all traffic is untrusted and that all resources may have been compromised, so everything in your network must be protected individually. The idea is similar to secure office buildings that have gate security to enter the building, security controls to access each floor, closed-circuit cameras

in all hallways, card locks on all doors, keys or access codes to safes, computers or even lockers, and so on, never assuming that previous security measures were successful.

The zero trust principle also can be expressed as "never trust, always verify", meaning you always must assume the worst in regard to security and, particularly, to internal threats. (For an extreme case of risks involving trust, see the "Trust No One" sidebar.) It all boils down to these three rules:

1. All access to internal or external resources must be secured regardless of resource location or traffic origin. All connections are to be treated as if they were external to the data center. Normally, internal users' connections are subject to fewer restrictions and controls. This usually can be done with encrypted tunnels—a well known, fairly standard and widely available security measure, which should prove easy (in principle) to apply to any network.

2. All users are subject to "need-to-know" restrictive access strategies, so they are allowed usage only of specific resources, depending upon their functions within the company. This is an application of the "least privilege principle" (which I describe later).

3. All network traffic must be logged and inspected. Logging (a passive solution, good for later forensic analysis) is usually done in systems, but real-time inspection (an active watch guard that prevents problems in the present) is more rare.

Least Privilege: This is probably the oldest principle that I cover here. It originated in the 1970s in the Department of Defense (DOD). Despite its age, it still is a basic and important design consideration, not only from a security point of view but also in terms of fault tolerance (dealing with failures in components or systems).

The principle of least privilege (or "least authority"), basically implies that a process or user should have only the authority (privileges, clearance) it needs to do its job without hindrances and nothing more—in other words, "barely there" permissions.

There are two corollaries to this rule:

- Default privilege: the default access rule should be "zero access"—total lack of access.

- Privilege bracketing: if temporary access is needed and granted, it should be rescinded immediately after usage.

Restrictions against system-wide actions make it harder to exploit vulnerabilities. Deployment (the fewer privileges an application needs, the simpler its deployment) and stability (if a program is limited as to the changes it can produce on a system, the possibility of negative side effects on other programs is lower) also are positively affected by this principle.

One standard technology, supported by network access control and infrastructure software, is role-based access control (which I mentioned earlier). To recap, first roles

are defined for different job functions, and required permissions are assigned to them.

RBAC goes beyond classic ACLs (access control lists) insofar as it allows for a finer-grained approach. For example, an ACL might allow or disable access to a given system file, but it won't limit the ways that file could be changed by the user. On the one hand, with RBAC you can define operations, such as "create account" or "change address", and users would be more constrained as to what they actually can do. On the other hand, finer-grained also means more roles to manage—so you win some, and you lose some.

Implementing true least privilege policies isn't that simple, however. Defining fine-grained policies (so processes will get the most minimum privileges they require) is complicated, because some needs may be defined dynamically, so more lax policies may end up being implemented. A complete implementation plan includes the following steps:

- Compile information on all resources, including software (systems, databases) and hardware (computer equipment, communications equipment) to define appropriate levels of security.

- Define a list of roles and sub-roles, based upon job functions and specify needed access rules for each role. This task should involve all stakeholders to articulate what's needed; trying to do it centrally from your IT department usually ends in failure.

- Incorporate RBAC across all systems, including internally developed ones, commercial applications, legacy systems and so on, and test it.

- Do periodic reviews of all of the above. An annual review is appropriate, and all jobs, functions, roles and permissions should be checked.

Real-Time Visibility: Attacks always happen in real time, and that makes real-time security a basic need. Your tools must be proactive and help stop incidents, rather than be reactive and just let you know after the fact.

The longer it takes your security team to detect a breach, the longer the malware has to take advantage of it, so it pays to notice red flags quickly and have enough information to be able to act appropriately. You must be alerted to suspicious events on the spot and monitor all kinds of events, including these among others:

- System resources and indicators, such as RAM usage, CPU load or running processes, that can let you know about intruder malware at work.

- Network-related events, such as failed access attempts, DNS queries and unusual end-point connections, that can point to malware infection or breach attempts.

- Critical (possibly system) files and other configuration changes, as unexpected changes may signal a rootkit.

- Arrival (and execution) of executable files, paying particular attention to illogical extensions: computer graphics files shouldn't be executable, and office suites shouldn't be spawning external processes.

- Sensitive data (files, databases) "in motion", possibly signaling a running data theft attack.

- USB devices, which can be used as entry points, as key loggers or as a vector for many other malware types.

To complicate things further, not every anomaly means an actual security threat or breach. As a simple example, a rash of uncommon logins may be related to salespeople using CRM software to plan their monthly schedule, according to a particular company sales cycle. Thus, continuous logging analysis must be flexible enough and allow fine-tuning, so you won't be swamped with false positives.

Ensured Compliance: This principle entails being able to prove you are complying with any and all relevant regulations. It can be quite costly if not fulfilled appropriately, for it can cause fines or legal costs, separate and apart from any damages malware might cause.

Companies must comply with several mandatory regulations (legislation, federal and local standards, contracts and so forth), which inevitably imply producing appropriate trails for internal and external audits; see Table 3.

TABLE 3. Common Security Recommendations/Regulations

TITLE	DESCRIPTION
BASEL III and GLBA	For financial institutions, related to matters like confidentiality and integrity of personal information, integrity of transmitted information and more.
HIPAA	For the health-care industry, related to confidentiality, integrity and availability of health-care information.
PCI	For merchants, regarding credit-card information.
Sarbanes-Oxley	For all publicly traded corporations, regarding integrity and privacy of financial data.
California S.B. 1386	For all organizations doing business in California, regarding confidentiality of customers' data.

No matter what anti-malware tools you deploy, you must be able to ensure that adequate logs are produced and that they satisfy the different external entities' requirements. (This is also a point in favor of automated tools, rather than human-dependent measures, which aren't likely to produce sufficient documentation.)

Modern Security Requirements

There is no single "silver bullet" for security, and all you can do is put a series of obstacles in you attackers' way to make entrance as complicated as possible and, should that fail, to make it hard for them to exploit the results of the break in. This multilayered approach is a good thing. If you depend on a single defense, attackers can

Trust No One

Open-source software usually is assumed to be better in security terms, because of the "many eyes" concept—that given enough people examining a piece of software, all bugs are trivial, and all problems can be detected and corrected.

Ken Thompson, one of the creators of UNIX, in his "Trusting Trust" address given upon receiving the ACM's Turing Award, explained how to write a compiler that would be able to plant any specific, desired trojan horse code in any or all compiled programs (for example, the "login" tool) and showed how the compiler itself could be hacked so no users ever would become aware of the modification, even having access to the original source code—neat and evil!

Check out Thompson's talk at http://cm.bell-labs.com/who/ken/trust.html; it's certainly worth a read.

concentrate on it, and after gaining access, everything in your servers will be wide open.

For the most solid protection, you need to consider both defense and monitoring, in case your defense didn't work.

Real-Time Risk Management: As I've mentioned, the security-threat landscape is changing and evolving constantly, so your security practices should be able to deal with such changes. Enterprise systems also are changing rapidly, impelled by many drivers: mobility, cloud computing, services, virtualization and containers, the Web and more. New environments plus new threats

make a volatile mixture. The ESG (Enterprise Strategy Group, http://www.esg-global.com) defined real-time risk management (RTRM) as a practice based on these three principles:

- Instantaneous knowledge: real-time information on asset changes, vulnerability assessments and threat data is required, so you can act as soon as possible.

- Comprehensive visibility: information should be wide in coverage and take into account all existing vulnerabilities.

- Constant assessment and adjustment of controls: security isn't a "set-and-forget" feature; rather, controls need periodic revision in order to ensure that they work adequately.

Achieving these goals requires event recognition capabilities able to detect sophisticated threats, high-level network monitoring that goes beyond specific hot-spot detection, threat monitoring intelligence to keep up with all changes in the security and malware landscape, and a high level of automation to ensure full, consistent, 24/7 monitoring.

Antivirus Software: As I mentioned earlier, this is essentially an after-the-fact tool that works only after a given threat has been found and analyzed. This automatically imposes two restrictions: first, only malware

that has appeared already (and in enough places) will fall under the purview of the antivirus companies, and second, as Windows is more prone to infections, it's possible that said companies will focus less on Linux servers, delaying possible solutions.

But, there is an even worse problem. If you are considering zero-day threats or attacks specifically geared at your servers, you'll be completely out of luck. Malware may succeed because common tools won't have a chance of recognizing it, as they aren't aware of the proper "signatures" they should look for.

Separate Workloads: "Workload isolation" is a concept that requires separating the different tasks on your system, each in its own server (compartment), which runs a software stack appropriate only for its (limited) objectives. Doing this with actual, physical servers would be highly cost-ineffective, so virtual machines and containers (VMware and VirtualBox, or Docker) are usually chosen to share physical hardware. Because each compartment is geared to a single task, it's more likely that you'll be able to lock it down more than a general-purpose server—fewer users, fewer privileges, fewer connections, fewer open ports and fewer running services imply a smaller attack surface and fewer vectors of attack.

Security advantages for this are obvious. If an attack succeeds, malware will manage to infect a virtual server (or container) but will have problems spreading to other servers, either within or without the same physical hardware.

Network Isolation: Someone once said that the only way to secure a server was to disconnect it from the network, turn it off and lock it in a safebox. Without going to such extremes, if your network is made up of separate, independent, mutually exclusive subnets, all possible malware attacks will be limited to a single portion of your network, leaving most of of your infrastructure unscathed.

Patching Policies: Given that most weaknesses derive from vulnerabilities in system software, you could argue that keeping your system up to date, with all needed software updates, is possibly the most necessary security measure to apply. Of course, you can't just blindly install updates from any kind of site, or you may end up doing even more harm. Also, you should have a careful policy as to which packages to update. Most Linux distributions recognize two kinds of upgrades: security updates (which may be downloaded and applied automatically) and everything else (requiring specific approval before installation). This means you could miss meaningful (to you) patches if they aren't considered a serious security threat (for most users), so you need to be alert for all possible required updates.

On the other hand, sometimes you need cutting-edge packages and may want to work with the latest possible code, directly downloaded from the developer's own site. This can be good in some ways (such as quickly getting rid of certain defects), but it also multiplies security risks. With an "accept everything as soon as possible" policy, you may be getting packages with new,

undetected problems. On production servers, you have to walk a thin line between heavily tested, older versions (which may have old bugs that attackers could take advantage of) and brand-new, latest-version packages (which could have new undiscovered bugs).

To sum it all up, a good patch management policy should include:

- A general hardware and software inventory, including version numbers for all software packages in use, that is kept up to date. This inventory can be correlated with discovered vulnerabilities to help program patching activities.

- The usage of standardized configurations to simplify testing patches and updating software.

- A process for testing patches, before general application, to check whether they will affect normal operations.

- A process for deployment and verification of patches, which may include specific backups to allow restoring systems to a previous state if the patches produce unexpected effects.

Application Whitelisting: The general idea for application whitelisting is related to the zero trust principle. Instead of trying to block suspicious software but letting everything else run by default, you opt for blocking all software by default and just allow specific programs to run.

Of course, this isn't quite so simple. Users, as it often happens, can be a challenge, because they are used to controlling their own PCs. Limiting their ability to make changes or run new software can be a cultural problem, but for more tightly controlled systems, such as servers, and fixed-function devices, like point-of-sale, implementing an application can be a much simpler process. Given the sensitive information often contained on these systems, having a highly effective prevention mechanism, such as application whitelisting, can make a lot of sense.

Taking all of this into account, even if an organization can't opt for a full lockdown policy, it still can benefit from the less rigid prevention and detection capabilities and real-time monitoring of file inventory, file executions and registry changes provided by most modern application whitelisting solutions.

Conclusion

The malware threat is growing with no signs of abating, and it's a given that all servers eventually will come under attack. Classic, outdated defenses aren't enough to deal with this threat, but several tools and methods can alleviate the problems and mitigate their impact. What really matters is taking them into consideration, implementing them as part of your security plans, and periodically re-evaluating them in order to test their behavior and incorporate possible enhancements.

The bottom line is security is a 24/7 job. Your servers quite likely already are or soon will be under attack. You need to implement malware defenses now.■