

Hidden Vulnerability Report Attack

An exploration of the risks associated with untrusted software

Aidan Mcpherson

Clemson University School of Computing
atmcphe@clemson.edu

Kalyaan Narnamapuram

Clemson University School of Computing
knarnam@clemson.edu

Laura Morgan

Clemson University School of Computing
lbmorga@clemson.edu

Owen Sullivan

Clemson University School of Computing
opsulli@clemson.edu

ABSTRACT

In this paper, we present a simple command-line utility hiding a system vulnerability detection script that reports potential kernel vulnerabilities on a Linux system to an external listening server. This open-source package, referred to as *penguinsay* with major component *vulnreport* and distributed publicly on GitHub ([permalink](#)), is used to demonstrate the security risks associated with installing software from unknown or unreliable sources.

We were able to successfully demonstrate the process by which an unsuspecting user installs an unrecognized software package and unintentionally installs a vulnerability reporting program alongside what they believe is a safe command-line utility.

1 INTRODUCTION

The average user's increasing use of and reliance on third-party software applications in the Information Age, while often necessary to accomplish complex work and personal tasks, can also be the source of major security risks. To combat this, a variety of vulnerability detection, monitoring, and reporting tools have been developed for use by system administrators to identify and mitigate potential risks. And, although most trusted software is now distributed via well-recognized means such as Ubuntu's "Universe" repository for free and open-source software packages, there is still some software being developed and distributed through project websites and other unofficial means (including peer-to-peer distribution).

If a bad actor can convince an unassuming user to install a seemingly harmless software package, including open-source, they are immediately empowered to bundle malicious scripts and programs that install and run completely transparently to that user. Our team's *penguinsay* package serves as a proof of this concept, containing a malicious program and associated daemon service. The malicious reporting tool *vulnreport* reads and reports system vulnerabilities to a lightweight external listening server.

2 BACKGROUND

Covertly hiding basic malware alongside normal, non-malicious command-line utilities has become simple for bad actors targeting Linux systems. To explore how this is possible, one must understand a few basic concepts:

The Debian package format used by Debian-based Linux flavors, such as Ubuntu, is a standardized format for software distribution that "contain[s] all of the files necessary to implement a set of related commands and features" [1]. This format, which is identified by the *.deb* file extension, allows for a number of executable scripts,

such as *preinst* and *postinst*, to be packaged along with compiled software and run "automatically... before or after a package is installed or removed" [1], thus allowing bad actors to download and execute malicious code without notifying users who install the package. These scripts can use tools like *wget* to download publicly-available tools like *linux-exploit-suggester*, which "is designed to assist in detecting security deficiencies for a given Linux kernel/Linux-based machine" [3] but can nevertheless be used for nefarious purposes.

The recent emergence of programming languages like Go, which has become known for its "fast compile times to build programs that start quickly and run on any system" [2], makes it even easier for bad actors to quickly develop and run listening servers for basic data transmission. Once a bad actor develops a simple pair of programs (a client and server) to handle the transmission and reception of reported vulnerability data, it becomes trivial to package these malicious programs in the Debian package format like a modern-day Trojan horse and trick users into installing them.

```
1 func handleConnection(conn net.Conn) {
2     defer conn.Close()
3
4     // Get client IP address
5     addr := conn.RemoteAddr().String()
6     fmt.Println("Receiving data from " + addr + "...")
7
8     data := *new(string)
9     // Read received data until EOF is reached
10    read := bufio.NewReader(conn)
11    for {
12        msg, err := read.ReadString('\n')
13        if err == io.EOF {
14            data += msg
15            go saveReport(addr, data)
16            return
17        } else if err != nil {
18            log.Fatal(err)
19            continue
20        }
21
22        // Add buffer contents to received data
23        data += msg
24    }
25 }
```

Figure 1: Receiving data from a client in Go

Although the Linux platform is often considered secure due to the prevalence of open-source software and the communities that audit and support them, attackers still have many avenues for deceiving victims. This is especially true in cases where users, many of whom implicitly trust open-source software, do not carefully inspect the software they download and install or the locations they obtain it from.

3 OBJECTIVES

The penguinsay project aims to demonstrate the ease with which attackers can conceal malware within a software package that appears "normal" to the average user by revealing the problems associated with installing unverified packages.

The goal of the `penguinsay` command-line utility is to present users with the Linux mascot, Tux, saying facts about the Linux operating system in ASCII form without revealing the malicious nature of the `vulnreport` background service.

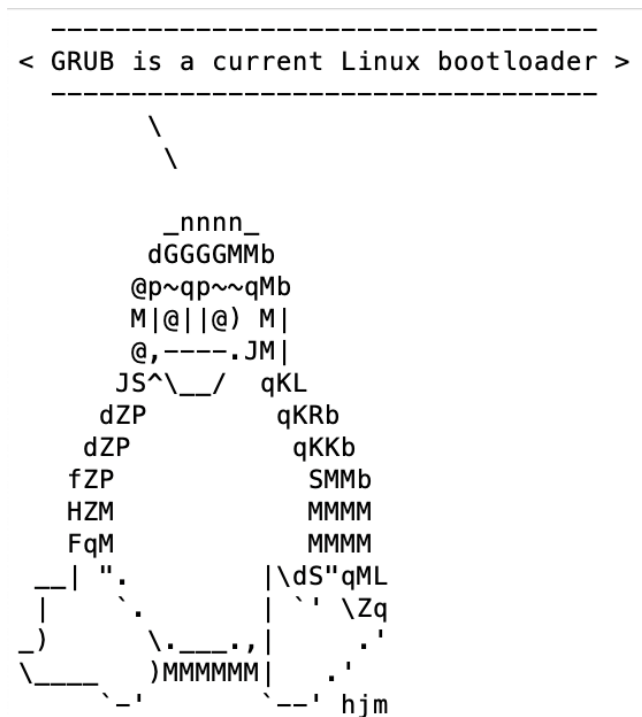


Figure 2: penguinsay running in a user's terminal

Meanwhile, the malicious `vulnreport` component’s goal is to leverage the open-source `linux-exploit-suggester` (referred to as “LES”) script to scan the system for kernel vulnerabilities and send them to an external listening server, all without logging any transmitted data in a way that a user could easily observe.

Neither the `penguinsay` package nor `vulnreport` attempts to hide the malicious scripts beyond simply suppressing command-line output during package installation and removal, as the primary objective of this project is to show how simple a malicious package can be. The ultimate objective is to bring attention to the importance

of understanding the components present in a software package, especially in the context of open-source software.

4 METHODOLOGY

To create a malicious Debian package, our team first studied the Debian package format and devised a structure for the execution of scripts which download and install LES. This required writing a series of shell scripts (`preinst`, `postinst`, `prerm`, and `postrm`) and redirecting the output of commands within them (which run with root permissions when a package is installed) to hide command output during installation. Since LES would need to be executed multiple times on a victim's system, the scripts place it under the `/usr/shared` directory.

Next, our team wrote a client-server pair of applications called `vrclient` and `vrserver`, together referred to as `vulnreport`, in Go. When compiled, `vrclient` is bundled into the package using a directory structure matching Linux's default directory for user-installed applications. A Linux daemon file was also created to enable automatic execution of the `vrclient` binary on a target system. The `vrserver` binary is used on a separate Linux server to act as the listener for reported data from the client application.

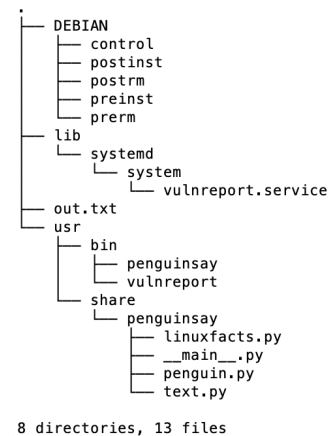


Figure 3: Directory structure of the penguinsay package

Finally, our team developed a simple command-line utility to serve as a front for the package. When a user installs `penguinsay`, they are able to execute it directly from their terminal. All the while, `vulnreport` is running in the background and reporting potential kernel vulnerabilities every time the user's system boots.

To ensure no traces of vulnreport are left behind when a user uninstalls penguinsay, Debian package scripts were leveraged to remove LES automatically.

5 ANALYSIS AND RESULTS

The result of a user installing penguinsay is that the user is able to access the penguinsay binary as normal while vulnreport sends potential vulnerability data to an external server. Vulnerability data, which is simply the output of the linux-exploit-suggester script, is saved as a text file on an external listening server (using the client's IP address and the current timestamp for the file name).

```

Mainline kernel protection mechanisms:

[ Disabled ] Kernel Page Table Isolation (PTI) support
              https://github.com/mzet-/les-res/blob/master/features/pti.md

[ Enabled ] GCC stack protector support (CONFIG_HAVE_STACKPROTECTOR)
              https://github.com/mzet-/les-res/blob/master/features/stackprotect
or-regular.md

[ Enabled ] GCC stack protector STRONG support (CONFIG_STACKPROTECTOR_STRONG)
              https://github.com/mzet-/les-res/blob/master/features/stackprotect
or-strong.md

[ Enabled ] Low address space to protect from user allocation (CONFIG_DEFAULT_
MMAP_MIN_ADDR)
              https://github.com/mzet-/les-res/blob/master/features/mmap_min_add
r.md

[ Enabled ] Prevent users from using ptrace to examine the memory and state of
their processes (CONFIG_SECURITY_YAMA)
              https://github.com/mzet-/les-res/blob/master/features/yama_ptrace_
scope.md

[ Enabled ] Restrict unprivileged access to kernel syslog (CONFIG_SECURITY_DME

```

Figure 4: A sample run of linux-exploit-suggester

This vulnerability data can then be used by an attacker to target the client's system with deeper, destructive attacks, as it exposes the attack surfaces available for exploitation. Immediately, it's clear that penguinsay's simple attack results in information about a victim's system being transmitted in plain text over the internet.

6 CONCLUSIONS

The penguinsay project highlights some of the critical security risks associated with the often unchecked trust and accessibility

afforded by open-source software. Bad actors can take advantage of users' often implicit trust in open-source to build and distribute simple, seemingly legitimate yet malicious applications. Our project demonstrates the ease with which malicious scripts can detect and report vulnerabilities on a system once a user has bypassed basic security features to install what they believe is safe software, and it further emphasizes the need for increased scrutiny of distributable software packages.

7 REFERENCES

- [1] Google. Command-line Interfaces (CLIs). The Go Programming Language. Retrieved November 11, 2024 from <https://go.dev/solutions/clis>
- [2] Software in the Public Interest. 2024. Chapter 7. Basics of the Debian package management system. The Debian GNU/Linux FAQ. Retrieved November 11, 2024 from <https://www.debian.org/doc/manuals/debian-faq/pkg-basics.en.html>
- [3] The-Z-Labs. Linux privilege escalation auditing tool. GitHub. Retrieved November 4, 2024 from <https://github.com/The-Z-Labs/linux-exploit-suggester>