My name is Robert Comella. I am a SANS Technology Institute (STI) student who is nearly finished with my master of science in engineering degree. When I am not speaking, writing and studying IT Security for the STI, I speak, write and study IT security on behalf of my clients in the Pittsburgh area.

I started my professional IT career as an intern 13 years ago but I have been building, programming, and playing with computers since the 80’s when my parents bought me a Commodore 64. Five years ago I took my first SANS class and focused my career on IT security which is fascinating to me.

I contributed the ROBAM (Read Only Bootable Alternative Media) and some other material to the “Protecting your Business from On-line Fraud” paper, many questions for the GCUX certification test, and I lead SANS mentor sessions in the Pittsburgh area.
Today I am going to tell you about my solution to a problem I came across while working with apt based Linux distributions. The process of eliminating unnecessary packages is important for several reasons which we will discuss. Unfortunately, it is an error prone and long process which many administrators skip. The next part of the presentation is my introduction of GUPI, the tool I used to make the process more bearable and hopefully get some more administrators to take the time to optimize their servers.
When an administrator builds a Linux server, they make many decisions. One of the most deceptively difficult ones is deciding which packages to install. Linux distributions try to pass package selection off as an easy choice. The administrator must simply choose a function they desire from the list the install program gives them…and poof! After much grinding, all the necessary packages install automatically. Luckily, the installation usually works and the resulting machine performs well enough to satisfy the one who installed it. Administrators focused only on functionality consider themselves finished and move on to the next task.

While it is impressive that Linux distribution masters can make this process as easy as it is, it is not possible for them to customize the system to the exact needs of every organization. The systems that result from these default installations, while completely functional, usually contain more software than is necessary for the assigned task to be completed. Since each software package is a potential source of vulnerabilities, the more installed packages there are, the more vulnerable the machine is. Secondly, unnecessary services and software take up precious resources, which leave fewer resources for the server’s assigned tasks. Finally, servers with more packages than necessary are more complex, which makes them more difficult to maintain. (Pomeranz, 2009)

Clearly it is desirable to create servers with only the software necessary to perform their assigned task. So it is up to the administrator to review all of the installed packages and their dependencies, figure out which ones can be removed and remove them. Then it is necessary to test the server to make sure everything still works. He/she must repeat the process until sure that all extra packages are gone. Easy right?

Removing Extra Packages

• Why?
  – Simpler to maintain
  – Utilize Fewer Server Resources
  – More Secure

• How?
  – Review Dependencies
  – Remove unnecessary packages
  – Test & Repeat

(Pomeranz, 2009)
Packages keep track of nine different types of dependencies.

- **Pre-Depends**: The dependent package must be installed and configured before the target package can be installed.
- **Depends**: The dependent package must be installed for the target package to work.
- **Recommends**: This relationship defines a strong but not absolutely necessary connection.
- **Suggests**: This dependency tells the installer that another package will improve the functionality of the target package.
- **Enhances**: This is similar to “Suggests” but from the opposite direction.
- **Breaks**: Installing the package will cause another package to stop functioning. Usually it is a version issue.
- **Conflicts**: Packages that conflict may not coexist on a machine.
- **Replaces**: Like it sounds, it is a direct replacement for another package.
- **Provides**: The installed package provides the functionality of the target package.

The result of these relationships creates a very onion-like structure where packages are layered on top of one another. Just as someone can peel the layers off an onion, revealing the layers below, administrators can peel the packages on the outside off, revealing other removable packages.

Which programs are on the outside of the onion? They are the packages upon which no other packages rely. To find them is a two-step process. First, make a list of the dependencies of all the packages installed on the system. Second, compare the list of dependencies to the list of installed packages. Any installed packages not on the dependency list are not necessary, and therefore, removable. The administrator must then decide if each removable package performs a service they require. If not, the packages are removed and the dependency list recalculated.
Gremlin’s Unnecessary Package Identifier, or GUPI, for short, automates the process of the identification of nonessential packages discussed in the prior section. When run, GUPI creates a list of all installed software. Then it creates a list of all the packages upon which they depend. Finally, it compares the lists and displays the packages it finds on the installed packages list but not on the dependency list. The user can then mark the packages “remove” or “keep”. When the user is finished with the list, GUPI will recalculate the package list and display a new list. Eventually, when nothing appears on the removable list, the user can ask GUPI to produce a script that will remove all the packages indicated while running the software.

Installing GUPI is easy. GUPI consists of four script files. The user must create these four files on the machine or copy them to it. All the files must be in the same directory. The only prerequisites known are apt-cache, dpkg, bash, and grep. Running the software is also simple. The user must 3cd to the directory where all the files reside and run the gupi.sh script. ./gupi.sh or, if the executable bit is not set for the package, bash gupi.sh. For a moment, the program shows the title screen. Next, it displays the main menu.
At first, the main menu is quite short, only showing the first few options. Once the user creates the package lists, the rest of the options will become available.

“Update the package lists from the Internet” will run `sudo apt-get update` to get the latest package information. Ubuntu will ask you to enter your password.

“Build working databases from Scratch (slow)” will build the database of packages. This process takes longer as the number of packages on the system increases. On slow computers with many packages, this process may take up to 15 minutes.

“Load Saved Data (fast)” will load saved database files so work that previously saved work can be continued. The saved file must reside in the same directory as GUPI. “Review Removable Packages” shows the list of packages it considers removable.

“Review Removable Packages” shows the list of packages that the user may remove.

“Review Kept Packages” shows the list of packages the user marked as necessary even though they could remove them.

“Review Removed Packages” shows the list of packages that the user tagged to remove. This allows the user to change his/her mind.

“Review Necessary Packages” shows the list of packages that cannot be removed because another package depends on them in some way.

“Review All Packages” shows the list of all packages the program knows about.

“Create apt Command” creates a script and places it in the directory with the program files. The script contains an `apt-get` command that will remove all the packages in the “removed package” list.

“Save Current Progress” saves the current working database for so it can be loaded later.

“Quit the Program” exits the programming unloading and wiping the working files from
When you chose any of the review options, the program will present you with the screen above. The package selection screen shows packages (sorted according to the review function you selected). The package list screen displays information about the current state of each package.

**Package Name:** Package name is the name given by apt to the package.

**Cur State:** This represents the Current State of a package. “I”, for “installed” is the only value.

**Fut State:** Future state is the state of the package after running the script generated by GUPI. It has two possible values “I” stands for “Installed”, and “U”, for “Uninstalled”.

**Reqd?:** The required field depicts whether or not another package depends on this one. Its possible values are “YES” or “NO!”

**Keep?:** The keep column tracks packages that the user marked to keep. Its possible values are “YES” or “NO!”

If the user wishes to change the state of a package, the user must type the number that is to the left of the name to bring up a package detail screen. We will cover that in a later slide.

Due to size limitations, GUPI only shows ten packages at a time. It is common, however, to have many more than ten packages in a particular list. The user can use “n” to see the next group of ten and “b” to go back to the previous list.

When GUPI displays the package list screen it makes a copy of the dataset in memory. The last three options allow the user to interact with that copy and the original dataset.

s: Will save the changes made to the states of the packages to the main database. While this will cause the dependency package information to be updated, it will not make changes to your system.
Choosing any of the numbers next to the listed packages will open the package modification screen. From this screen, the user is able to control the various states of the package if permitted. The screen shows the package’s current states at the top just like on the package selection screen. Below it shows apt’s short package description.

- **f**: toggles the future state of the package
- **k**: toggles the keep state of the package
- **m**: attempts to display the man page for the package. If there is no man page, it will display a message telling the user of its failed attempt
- **s**: accepts the changes and returns the user to the package list screen
- **q**: rejects the changes and returns the user to the package list screen
Two special cases of the package modification screen exist. The first special case occurs when the package is one of several that will fulfill a dependency. In other words, the package currently installed or the one listed next to it fulfill a dependency. GUPI does not have the capability to exchange them within the software, but this feature will at least inform the user that they have a choice of which to install. Sometimes the alternate package will be smaller or cover a more appropriate group of features which make it more attractive to the administrator.
The second and more common special case occurs when one package provides another. Due to the way GUPI calculates removable packages, it is unable to take into account for the “Provides” dependency type en masse. The workaround is to check each package when this screen appears. If any package provides another, the original package, if not called directly by the dependency file, will show as removable. When this screen loads, GUPI will check the names of all the packages that this one provides against the dependency list. If it finds a match then GUPI informs the user and disables the “i” option. If the package is not on the list then the “Provides” information is shown, but no other changes occur.
In order to build a server that has as little software on it as possible, I find it better to start from nothing and only install those programs necessary. There are Linux distributions that really exemplify this. The user can start with an entirely blank hard drive and load source code for each program then compile everything from scratch. While I found this exercise incredibly educational and interesting, most clients want results that do not take hours or days to compile.

The idea of a null box is the happy medium between starting from a blank hard drive and compiling everything, or simply using the install wizard to install a new server. The null box is a standard minimal install of a Linux distribution (in this case Ubuntu) where the administrator removes all but the most basic functionality. What the basic functionality is will change between environments, but in this instance, all machines must at least do the following:

- Boot and communicate with network
- Update installed packages
- Install new packages

While the Null Box is lean, it is not necessarily secure. There is no security software installed on the machine at all. Placing such a machine on a production network, especially one connected to the internet, is dangerous. There are however, many choices when it comes to security software, all of which depend on company policy and server function. It is important for the system builder to choose which is best.
It may be apparent now that the idea for the null box came first and GUPI was the tool created to build it. Using GUPI in the creation of the null box is relatively simple. Install the most basic version of Ubuntu server and place GUPI in the home directory. Run GUPI and remove all packages that do not affect the three criteria. Repeat the process until removing any packages will prevent the computer from accomplishing one of its three stated functions.

There are some gotchas. First, there are some packages that have circular dependencies. That means each lists the other in their respective dependency list. According to GUPI these packages can never be removed. In reality, they can be if it is done in pairs. There are three examples of these packages in Ubuntu.

- tasksel and tasksel-data
- perl and perl-modules
- kbd and console-setup
While some packages are technically removable, their absence causes issues on the system. Removing apt-utils prevents apt-get from setting up packages. In some cases, this is not a problem because the packages are very simple. The dhcp3-client package obtains IP address information during the boot process. Assigning a static IP address before removing this package will avoid any problems. It is possible to remove grub-pc. Doing so prevents the system from updating the boot files. If during the uninstallation the administrator chooses to remove the files from the /boot directory, the computer will be unable to boot. Removing iptables will prevent the user from modifying the kernel side of the firewall. This action effectively disables the firewall because turning it on is a modification. The program that builds the message of the day file (MODT) uses the lsb-release package. The administrator can fix the non-critical error by modifying /etc/update-motd.d/00-header file. Certain packages require ncurses-base to install correctly. Removing it prevents the installation of those packages. There is nothing that prevents an administrator from removing ncurses-base after all necessary software is installed and configured. All in all, the system may be diminished without these packages but it will still fulfill its mandate.

The sudo package is a little different. In the Ubuntu distribution, the creators have decided that no one should ever log on as root. The Ubuntu designers gave the root account a password that users cannot enter, effectively disabling it. It is possible to remove sudo, but the administrator must take two actions first. First, the user must set the root account’s password to something known. The second step is to export the SUDO_FORCE_REMOVE system variable by typing “export SUDO_FORCE_REMOVE=yes”. Then it is possible to remove sudo and retain an
Some packages must be marked as keep in order to allow the server to fulfill its function of booting and installing packages. Removing apt, tar, or whiptail prevents the installation of new packages. apt downloads new packages, tar opens the tarball, the form in which they are stored, and whiptail is a hidden dependency of debconf. The package bash provides the command shell. The user can replace it with alternatives if they so desire but some sort of command line is necessary. debconf-i18n is necessary but debconf-english can replace it. debconf-english is slightly smaller because it contains only support for English-speaking users. Administrators can remove e2fsprogs but when they do, the boot scripts that check the drives fail. The package gpgv validates the packages in the Ubuntu package store. If the user removes it, the server complains that the package store is untrusted. gzip unzips packages retrieved from the package store, without it most packages cannot be installed since they cannot be unpacked. The administrator can also remove hostname but doing so will cause errors to occur each time the user runs a command in the shell. The command usually works but the server takes 10 to 15 seconds to time out while attempting to find the name of the host before continuing. The linux-image-2.6.XX-XX-generic-pae is the kernel. Removing this package prevents the computer from functioning. login provides the ability for users to log into the computer, removing it stops users from logging on. These are the “removable” packages which, if removed, will definitely prevent the server
Summary

This presentation showed:

- Why it is a good idea to eliminate as many packages as possible
- How GUPI makes it simpler to do so
- The null box
- The consequences of removing some packages