TROJANIZING THE AES CRACKER

RnD Report

Karan Ganju
Indian Institute of Technology Bombay
Department of Computer Science

Contents

Introduction

The aim for the RnD Project was to craft a feasible proof-of-concept attack for an unsuspecting victim exploiting the Linux processor cache to perform side-channel attacks on the victim's AES encryptions. For the attack to succeed, we would have to provide the Center for Advanced Analytics(CAA) two things described below

- All the network packets (encrypted using AES) exchanged between the victim and a particular server of the attacker's choice, say www.gmail.com
- The dump of the cache lines accessed throughout the communication

Not only must the malicious program obtain this information but it must also function without alerting or raising the suspicion of the victim and establish a connection to the attacker to relay this information back to the CAA.

The next section describes how the attack is launched from creation of the malicious trojan to the relay of sniffed data and cache dumps from the trojan on the victim to the attacker.

Attack Analysis

Creation of the Trojan Package

Propagating the malicious executable to the victim's machine is particularly tough in the case of Linux as files downloaded from the browser over the internet are not provided executable permissions. An alternative, easier and perhaps more flexible approach would be to instead integrate this program with a Debian package instead as shown in the Metasploit guide[2]. This involves the following steps.

1. **Choose a harmless Debian package**
   This package would serve to be the primary utility demanded by the victim which could be anything from a simple game like the minesweeper game (Freesweep) that I have used for the purpose of a demo to a complex web browser. Since we rely too highly on the user's naivete to download the package from an untrusted source, a more effective attack would require either a good amount of social engineering providing sufficient incentives to the victim to install the package from the alternative source or the Debian package to be not easily accessible, say, for example a Debian package providing custom Debian GUI themes.

2. **Extract the package and insert the relevant trojan files**
   The package can be extracted with the command given below.

   ```
   sudo dpkg -i <debian_package.deb> <extracted_folder>
   ```

   The trojan files can then be inserted into the extracted folder which will be unloaded into corresponding folders when the package is installed.

3. **Write a postinst and prerm script for the package**
   Both of these bash scripts are the drivers behind launching the malicious executable.

   postinst This script is called during dpkg configuration of the package which occurs after the relevant files of the package are unpacked. This is the core of the attack as this is the only time when we will receive sudo permissions without alerting the victim. Hence, it is our only window when we can utilize root permissions. The postinst bash script will be used by our trojan to launch the sniffer in the background (which necessarily requires root permissions to sniff packets).

   prerm This script is called just before the relevant files are deleted at the time of uninstallation of the program. Hence, this script can be used either to hide the malicious files which may have aroused the victim's suspicion leading him to uninstall the package, or as in our case, to clean the victim's machine of the malicious files.

4. **Build the Debian package**
   This will create final package which will be used to propagate the AES cracker to the victim's machine. This can be done with the command given below.

   ```
   sudo dpkg-deb --build <debian_package_folder>
   ```

To automate all of these steps, a make_deb.sh bash file was created, which downloads the freesweep package, extracts it, copies the sniffer and the espionage network code to the package, creates corresponding postinst and prerm scripts, and repackages the debian package. Certain parameters which have to be specified at the time of creating the package and cannot be changed afterwords are given below.

- Attacker's IP address and port to which the sniffed data and cache dump is relayed to
- Interface on which the sniffer sniffs packets and the IP address of the server whose communication with the victim is to be tracked

Network Sniffer

A sniffer program was written with the help of some online resources[3]. This sniffer starts in the background as soon as the package is installed. It uses the libpcap library and applies certain pcap filters which allow it to capture data passed between only the victim and the external server we had chosen. A loopback function dictates that the sniffer parses the packet data to obtain the TCP payload for all filtered packets, which is then written to a file named out_<ip of remote server>_<port of remote server>. As soon as the first payload is received, the sniffer program forks and the child is made to obtain the cache dump.

Obtaining Cache Dump

To obtain the cache dump, the program provided by Ravi was started by simply running the bash file run.sh. Since, analyzing the code for the Espionage Network was not within the scope of my work, being peripheral to it's functioning, I have taken it to function as a black box. However, this does result in certain syncronization issues which have been discussed in the Future works section.

Communication with CAA

As mentioned above, a child process is launched by the sniffer on receipt of the first payload. This child process then starts the espionage network. In order to detect the completion of the cache analysis by the spy controller, a SIGCHLD handler is set for the parent sniffer process. This handler is called whenever the child process dies, i.e., when the cache analysis is complete and cache dumps are ready. At this point, we can stop the sniffer and send both the sniffed data and the cache dump to the CAA or the attacker. A socket is opened on the victim's machine which is used to transmit both these files, sending the word "END" to mark the end of either file.

Miscellaneous

Code for a simple client and server was also written (mostly by Ravi and partly by me) so that the attack could be carried out on them for the purpose of a demo. This helps during debugging as well because the packets sent are simpler and less cluttered than the ones sent over the internet. That being said, the final aim is to target commercial servers instead.

Extensions

Hiding command from ps: The sniffer program changes the argv[0] argument so as to name itself [kworker/1:2], which is that of a kernel thread, a thread that not many general users care about or inspect. Secondly, it also plays around with the buffer provided by the kernel for these argv[] arguments so as to hide it's command line arguments from the ps command[4].

Future Work

- **Synchronization -** The Centre for Advanced Analytics requires that the cache dump and the sniffed data be exactly synchronized, i.e. the cache dump should exactly reflect the encryptions/decryptions of the sniffed data from the start. At the current point, the functioning of the espionage network is taken to be a black box. Accomplishing synchronization would require the sniffer and espionage network to function in a more integrated manner.
- **Polymorphism -** The malicious files within the package can be encrypted so that they are not revealed easily upon extraction of the package. They would then require a decryption engine to be sent along. Any common engine would only alert anti virus softwares so the best solution would be to use a custom one. As soon as the package is installed, the decryption engine could decrypt the malicious files and then delete itself.
- **Command and Control Centre -** A more hidden and advanced version of the threat would require some form of persistent communication with a control centre which would issue orders to start, stop, abort(and delete all malicious files), etc. In this case, the sniffer would only activate when the control centre orders it to, remaining inactive till then.

REFERENCES

1. **Design and Implementation of an Espionage Network for Cache- based Side Channel Attacks on AES**
   Bholanath Roy, Ravi Prakash Giri, Ashokkumar C., Bernard Menezes
   SECRYPT 2015: 441-447

2. **Offensive Security guide on creating Linux Trojans**
   https://www.offensive-security.com/metasploit-unleashed/binary-linux-trojan

3. **Packet Sniffer in C**
   http://simplestcodings.blogspot.in/2010/10/create-your-own-packet-sniffer-in-c.html

4. **Hiding arguments from the ps command**
   http://netsplit.com/hiding-arguments-from-ps