****

**EE6042- Host and Network Security**

**Group Project**

Due Date: Friday the 21st of April, 2017

Course: Information and Network Security MENG

Department: Electronic & Computer Engineering

Student: Hugh Mullins, Student ID: 16106989

Student: Sunday Jimoh, Student ID: 16090462

Student: Conor McDonnel, Student ID: 16120604

Student: Raksha Ramdas, Student ID: 16052439

Student: Samaa Alsafwani, Student ID: 16087941

**Table of Contents**

[1 Dirty Copy-on-Write (CVE-2016-5195) 4](#_Toc480210213)

[1.1 Introduction 4](#_Toc480210214)

[1.2 Impacts of Exploiting the Dirty COW Vulnerability 4](#_Toc480210215)

[1.3 How to execute a Dirty COW Exploit 5](#_Toc480210216)

**Table of Figures**

[Figure 1: Analysis of the impact of The Dirty COW exploit on a system 5](#_Toc480203924)

# Dirty Copy-on-Write (CVE-2016-5195)

## Introduction

Dirty copy-on-write (Dirty COW) is a security vulnerability present in the Linux kernel since version 2.6.22 which was released in September, 2007 and it has been exploited actively since October, 2016. Its CVE designation is CVE-2016-5195. It affects all major vendors of Linux including Ubuntu, Debian, Red Hat, Fedora and even Android, which is also powered by the Linux kernel. It was discovered by a Linux security researcher, Phil Oester.

Copy-on-write (COW), sometimes referred to as implicit sharing or shadowing, manages memory resources and allows for more than one process to share a page until a user writes to the page, this is known in programming as marking a page dirty.

Dirty COW is a local privilege escalation bug that exploits a race condition in mm/gup.c in the Linux kernel 2.x through 4.x before version 4.8.3, taking into consideration the way the Linux kernel’s memory subsystem handles the copy-on-write (COW) breakage of private read-only memory mappings. An unprivileged user or a local attacker could exploit this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system. This vulnerability could be used by the attacker to modify existing setuid files with instructions to gain administrative privileges. The exploitation of the Dirty COW bug leaves no traces of anything abnormal in the system logs.

## Impacts of Exploiting the Dirty COW Vulnerability

Some of the impacts of exploiting the Dirty COW bug on Linux based systems include:

* The flaw allows attackers with local system accounts to modify on-disk binaries, bypassing the standard permission mechanisms that would prevent modification without an appropriate permission set. This is achieved by the racing the madvise (MaADV\_DONTNEED) system call while having the page of the executable mmapped in memory.
* The Linux kernel contains many binaries which are read-only, and can only be modified or written to by a user of higher permissions, such as the root. When an attacker escalates system privileges using the Dirty COW exploit they can change files, such as /bin/bash, so that it performs additional, unexpected functions such as a key logger to steal passwords and other sensitive information from the legitimate users.
* Although it is a local privilege escalation bug, it can be used by remote attackers in conjunction with other known exploits that allow remote execution of non-privileged code to achieve remote root access on an affected computer. The attack itself does not leave any traces in the system log.
* It can also be used to obtain root permissions in any Android device up to Android version 7.

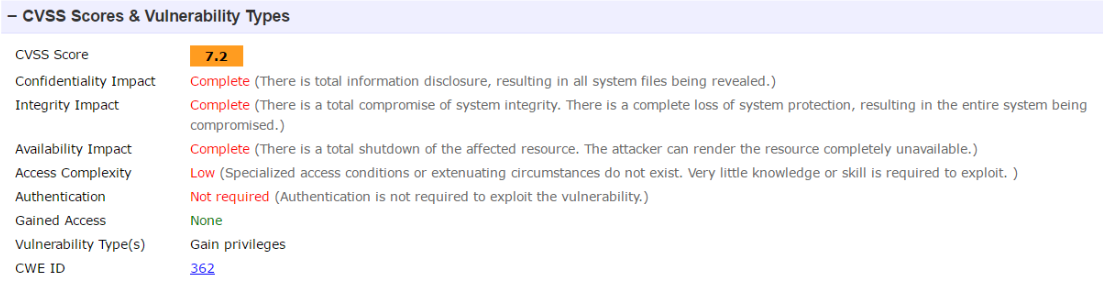


Figure 1: Analysis of the impact of The Dirty COW exploit on a system

## How to execute a Dirty COW Exploit

Some of the known techniques of exploiting the Dirty COW bug include:

* This first Dirty COW exploit seen in the wild takes two arguments a filename and a string to be written to the file.  The file specified is opened as read-only and is then mmap’ed into the current process read-only and with [MAP\_PRIVATE](http://man7.org/linux/man-pages/man2/mmap.2.html) specified to create a private COW mapping.  It then starts two threads. The first thread opens /proc/self/mem read-write and loops millions of times writing the specified data to the command line at the offset of the mmap’ed file. Writing to the /proc/self/mem triggers COWs, but since these pages are marked private, the memory pages written to will not be written back to the mmapped file. The second thread loops multiple times, calling the madvise system on the same offset w/ the flag MADV\_DONTNEED which tells the kernel to discard pages that are mmap’ed. It continues writing to memory and, at the same time, informs the kernel that memory is no longer needed thus exposing a race condition where changes to private pages could be propagated on the underlying file.
* Another exploit uses the pokemon exploit as a base and automatically generates a new passwd line. The original /etc/passwd is then backed up to /tmp/passwd.bak and overwritten with the new line. The user will be prompted for the new password when the binary is run. After running the exploit, you would be able to login with the newly created user.