FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Analýza bezpečnosti Linuxového clipboardu
Student:	Benjamín Peraus
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

1) Popište obecné bezpečnostní aspekty clipboardu. Zaměřte se zejména na jeho použití pro přenos hesla ze správce hesel do cílového programu.
2) Seznamte se s implementací clipboardu na systémech Linuxového typu.
3) Analyzujte jejich specifické bezpečnostní vlastnosti, formulujte slabá místa stávajících implementací.
4) Vyhodnoťte nalezené hrozby a navrhněte jejich řešení.
5) Implementujte proof-of-concept vašeho návrhu.
6) Diskutujte jeho silné a slabé stránky a možnosti nasazení.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 30. ledna 2022 v Praze.

Bakalářská práce

ANALÝZA BEZPEČNOSTI LINUXOVÉHO CLIPBOARDU

Benjamín Peraus

Fakulta informačních technologií
Katedra počítačových systémů
Vedoucí: Ing. Josef Kokeš
11. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat zejména panu Ing. Kokešovi za jeho výborné odborné vedení, cenné rady, čas v podobě konzulatcí a zodpovědný přístup. Dále bych rád poděkoval Fakultě informačních technologií ČVUT v Praze, díky které jsem získal dostatek vědomostí k tomu, abych mohl práci vytvořit. Poslední, komu bych chtěl poděkovat je moje rodina, díky které jsem měl velké množství podpory ve studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2022 .....

Abstrakt

Tato bakalářská práce se zabývá bezpečnostními aspekty clipboardu používaném v operačních systémech ke kopírování a vkládání obsahu ze zdrojové aplikace do cílové. V práci je podrobně zmapována aktuální situace, v jaké se nachází bezpečnost clipboardu. Zvláštní pozornost je věnována používání clipboardu k přenosu hesla ze správce hesel do cílového programu.

Podrobně jsou analyzovány bezpečností aspekty implementací clipboardu, které se týkají operačních systémů Linuxového typu. Závažnost nalezených hrozeb je ohodnocena metodikou CVSS. Pro tyto implementace je navrženo řešení, které by umožnilo bezpečnější práci s clipboardem.

V rámci této práce vznikl nástroj pro bezpečné kopírování, který přenáší data P2P pomocí clipboardu. Nástroj je implementovaný v jazyce C/C++ pro protokol Wayland a kompozitor Weston (nicméně nástroj je funkční i na kompozitoru KWin a měl by být funkční na všech konvenčních kompozitorech). Tento nástroj je proof-of-conceptem tohoto řešení a v práci je vysvětleno, jakým způsebem je ho možné zakomponovat do správce hesel. Silné a slabé stránky tohoto řešení/implementace jsou v této práci diskutovány, stejně tak možnosti nasazení jsou zde rozebrány.

Klíčová slova clipboard, Wayland, Weston kompozitor, správce hesel, Linux

Abstract

This bachelor thesis deals with the security aspects of the clipboard used in operating systems to copy and paste content from a source application to a target application. The thesis provides a detailed research of the current clipboard security situation. Particular attention is paid to the use of clipboard to transfer passwords from the password manager to the target program.

The security aspects of clipboard implementations as they relate to Linux-type operating systems are analyzed in detail. The severity of the threats found is rated using the CVSS methodology. For these implementations, a solution is proposed to make the clipboard more secure.

In this work, a secure copying tool that transfers P2P data using clipboard has been developed. The tool is implemented in C/C++ for the Wayland protocol and the Weston compositor (however, the tool is also functional on the KWin compositor and should be functional on all conventional compositors). This tool is a proof-of-concept of this solution and the thesis explains how it can be incorporated into the password manager. The strengths and weaknesses of this solution/implementation are discussed in this thesis, as well as deployment options.

Keywords clipboard, Wayland, Weston compositor, password manager, Linux

Seznam zkratek

ACL	Access Control List
API	Application Programming Interface
CLI	Command Line Interface
CVSS	Common Vulnerability Scoring System
FD	File Descriptor
GID	Group Identifier
GUI	Graphic User Interface
MIME	Multipurpose Internet Mail Extensions
MITM	Man-in-the-Middle
OS	Operating System
PID	Process Identifier
PNG	Portable Network Graphics
P2P	Peer-to-Peer
RFC	Request for Comments
STL	Standard Template Library
UI	User Interface
UID	User Identifier
URL	Uniform Resource Locator
UTI	Uniform Type Identifier

Úvod

Clipbord, česky často nazýván schránka, je jedna z utilit operačního systému, která uživatelům umožňuje kopírovat nějaký obsah z jedné aplikace do druhé. Je to velmi častá činnost každého uživatele bez ohledu na jeho pokročilost.

Základní myšlenka, která vedla k napsání této práce, byla, že uživatelé často používají clipboard na různě citlivá data (dokonce i hesla), aby je přenesli z jedné aplikace do druhé. Následně došlo k uvědomění si faktu, že clipboard je v podstatě sdílené médium, ke kterému mají přístup veškeré aplikace, a že zřejmě není v pořádku, aby se citlivé údaje dostaly mimo zdrojovou a cílovou aplikaci. Vzhledem k tomu, že se v historii ukázala důležitost segregace privátních dat od veřejných, což způsobilo změnu přístupu k programování, je dnes prioritou, aby se sdílelo co nejméně dat již na úrovni objektů. Pokud se podíváme na clipboard optikou dnešních programovacích přístupů, není zřejmě v pořádku, aby měl přístup k datům clipboardu každý proces. Tento pohled vedl k myšlence, zda vůbec existuje nějaká bezpečností politika u clipboardu, případně zda existuje alespoň možnost předávat data mezi dvěma aplikacemi bezpečně pomocí clipboardu.

U dnešních OS je clipboard velmi komplexní sada utilit. Nejedná se pouze o jednoduché předání dat z jedné aplikace do druhé, ale jsou zde funkcionality jako evidování historie výběrů předávaných pomocí clipboardu a možnost je opět vložit, uplatnit na ně nějaký regulární výraz nebo jinak zeditovat a mnoho dalších. S rostoucí použitelností tohoto nástroje roste i jeho vektor bezpečnostních hrozeb, kterým je nutné předcházet a které bychom se měli snažit eliminovat.

Zvláště důležitá motivace pro zabývání se bezpečnostními aspekty tohoto nástroje je také to, že je dnes možné, a tudíž používané, synchronizovat obsah clipboardu mezi více zařízeními nebo jej sdílet z virtuálních strojů.

Cíle bakalářské práce

Hlavním cílem této bakalářské práce je navrhnout řešení bezpečnostních hrozeb clipbordu, které jsou spojeny především s používáním clipboardu na přenášení citlivých dat, jako jsou například hesla. Funkčnost řešení bude ověřena implementací proof-of-conceptu v jazyce C/C++. Program bude reálně použitelný v operačních systémech Linuxového typu, které budou používat konkrétně tu implementaci clipboardu, se kterou je v programu počítáno. Také je nutností, aby cílové systémy měly k dispozici knihovny, které budou v implementaci používány.

Dalším velmi zásadním cílem této bakalářské práce je podrobně diskutovat, jaké jsou silné a slabé stránky mnou navržených řešení a na jakých prostředích operačních systémů Linuxového typu je možné tato řešení nasadit. Mechanismy, které budou zajišťovat bezpečnější fungování clipboardu, lze umístit na více míst, kde probíhá komunikace ohledně kopírování ze zdrojové aplikace do cílové. Každé místo, kde se mechanismus umístí, s sebou nese různá úskalí, a proto je velmi důležité tyto možnosti zanalyzovat a diskutovat.

K dosažení těchto cílů je nutné analyzovat obecné bezpečnostní aspekty clipboardu, podrobněji se seznámit s implementacemi clipboardu v systémech Linuxového typu a jednu z těchto implementací vybrat jako cílovou pro náš proof-of-concept.

Cílem práce není vytvořit nový clipboard, který by byl bezpečnější. Taktéž není cílem této práce naimplementovat nějaký komplexní systém spravující přístup ke clipboardu bezpečnějším způsobem, který by fungoval napříč různými prostředími. Půjde o vytvoření konceptu, který by mohl být nadstavbou jedné z aktuálních implementací clipboardu v OS Linuxového typu, která by zajistila bezpečnější chování clipboardu. Práce se nijak nezabývá mechanismem drag-and-drop, neboť cílem práce je dosáhnout bezpečného chování clipboardu, nikoliv nahradit clipboard za mechanismus drag-and-drop.

Kapitola 1

Clipboard a jeho funkce

Tato kapitola popisuje obecné principy fungování clipboardu napříč operačními systémy. Dále je zde popsáno, proč je výhodné používat formáty definující typ přenášených dat. Kapitola zavádí základní pojmy spojené s problematikou clipboardu, jako je například clipboard manager.

1.1 Clipboard

Clipboard, česky nazýván schránka, je nástroj OS, který slouží k jednoduchému předávání dat mezi aplikacemi. Existuje prakticky ve všech operačních systémech, které disponují nějakým GUI, dokonce existují jednoduché implementace pro OS, které mají pouze CLI. Napříč všemi konvenčními operačními systémy je základní princip tohoto nástroje stejný. Označíme nějaký obsah, který chceme zkopírovat nebo vyjmout. Následně pomocí vstupního zařízení vyvoláme událost kopírování, a obsah je zkopírován (respektive přesunut) z nějaké zdrojové aplikace do clipboardu. Následně vyvoláme, pomocí vstupního zařízení, událost vložení ve vhodném místě, a obsah clipboardu je přesunut do cílového místa v cílové aplikaci.

Existují různé implementace clipbordu, které mají různou úroveň pokročilosti a použitelnosti. Nejjednodušší implementace umí předávat pouze data ve formě textu. Sofistikovanější implementace dokáží pracovat i s daty, jako jsou obrázky nebo multimediální soubory. Jak roste pokročilost a použitelnost UI, tak roste také použitelnost a pokročilost clipboardu. Dnes už je clipboard ne jeden nástroj, ale spíše sada nástrojů, které tvoří jeden komplexní celek zaměřený na jeden účel. S rozvojem cloud computing byl clipboard povýšen na nástroj, který umožňuje kopírovat data nejen z jedné aplikace do druhé na úrovni jednoho zařízení, ale dokonce z jedné aplikace do druhé v rámci více oddělených zařízení, která jsou připojena k internetu.

Účelem clipboardu je zřejmě zjednodušit uživateli práci v operačním systému, konkrétně umožnit mu si dočasně „odložit“ nějaká data, která vzápětí znovu použije. Uživatel tedy nemusí znovu přepisovat například text, který již napsal. Má možnost ho nechat, pomocí jednoduché kombinace kláves, dočasně zapamatovat operačním systémem, a poté znovu použít. V některých OS se ještě více zaměřili na zjednodušení použití tohoto nástroje tak, aby plnil svůj účel. Vzhledem k tomu, že musíme text označit a ještě navíc vyvolat akci kopírování, je to zdánlivě zbytečná operace, protože tuto dvojici operací děláme společně automaticky. Proto existují implementace, které považují označení textu za operaci kopírovaní a operace vkládání je spojena s událostí zmáčknutí prostředního tlačítka myši. Tato možnost je sice příjemná, ale má svá bezpečností rizika, které si probereme v dalších sekcích.

1.2 Více clipboardů

Clipboardů může být v rámci jednoho operačního systému dokonce více. V běžné praxi se s tím můžeme setkat například u operačních systémů Linuxového typu. Jeden clipbord je komplexní nástroj se standardním ovládáním tak, jak jsme si popsali v sekci 1.1, a druhý clipboard je jednodušší varianta, která funguje pouze ke kopírování textu a zpravidla má zjednodušené ovládání (například pomocí označení a prostředního tlačítka myši, jak jsme si popsali v sekci 1.1). [1]

Další clipboardy se vyskytují například již lokálně na úrovni nějaké aplikace. Ty mohou být naprosto oddělené od systémových clipboardů. Jako příklad si můžeme uvést velmi známý editor Vim. [2]

1.3 Formát a obsah předávaných dat

Jak již bylo zmíněno na začátku kapitoly 1.1, dnešní implementace clipboardu umí pracovat s různým typem dat, například s multimediálními soubory. Podle typu obsahu, který chceme předat prostřednictvím clipboardu, je nutné přizpůsobit předávaná data a také vědět, jak tato data interpretovat. Pokud například budeme předávat obrázek ve formátu PNG a vložíme ho do nějakého textového procesoru, tak očekáváme, že se někde v našem textu objeví onen obrázek. Nicméně pokud by textový procesor (jakožto cílová aplikace, do které vkládáme obsah z clipboardu) interpretoval obsah z clipboardu například jako text, docílili bychom nechtěného efektu. Textový procesor by byl plný „nesmyslného“ textu.

Z těchto důvodů se musí při vkládání obsahu do clipboardu specifikovat, o jaký formát a obsah dat se jedná. Stejně tak cílová aplikace musí znát tato metadata. Proto existují různé standardy, které popisují formát a obsah těchto dat. Tyto standardy se liší napříč operačními systémy. V OS Linuxového typu se jedná především o MIME typy, Apple nyní používá UTI a v OS Windows Microsoft definoval typy s názvem standardní clipboardové formáty. [3][4][5]

Díky těmto metadatům je možné při kopírování nějakého multimediálního souboru (například obrázku ve formátu PNG) nekopírovat celý binární soubor, pokud to není třeba. Můžeme předat pouze cestu k souboru ve formě krátkého textu a cílová aplikace už si poradí s obrázkem podle vlastního uvážení.

1.4 Clipboard manager

Clipboard manager je nástroj, který se dnes považuje za součást clipboardu. Jeho implementace se liší v závislosti na implementaci clipboardu jako takového. Úkolem clipboard manageru je poskytnout funkcionality jako je evidování historie našich kopírovaných dat a možnost je opět kopírovat, editovat tato data, uplatnit na ně nějaký regulární výraz a další. I například sdílení clipboardu mezi více zařízení může být právě navázáno na tento nástroj. [6][7]

Z hlediska implementace se může jednat o obal nad standardním clipboardem, přes který k němu přistupujeme. Pokud budeme brát clipboard jako serverovou aplikaci, byl by tento typ clipboard manageru obal nad touto aplikací, tím pádem by se stále jednalo o serverovou aplikaci. Další možnost je, že se jedná o monitorovací aplikaci, která by byla hierarchicky na úrovni běžného klienta. Tento typ si popíšeme ještě podrobněji v kapitole 3. Tento nástroj může být již součástí implementace standardního clipbordu, čili se nemusí jednat o dva oddělené nástroje. [6][7]

Kapitola 2
Bezpečnost clipboardu

Tato kapitola se věnuje analýze obecných bezpečnostních aspektů clipboardu napříč různými operačními systémy. Jsou zde popsány aktuální bezpečností hrozby, které jsou nejfrekventovanější. Dále je zde rozebrána problematika předávání hesla ze správce hesel do cílové aplikace pomocí clipboardu.

2.1 Časté bezpečnostní hrozby

Clipboard vznikl jako jednoduchý nástroj na předávání dat dat mezi aplikacemi. Ve starších operačních systémech se jednalo pouze sdílenou část paměti napříč procesy. Tento nástroj nebyl navržen zřejmě už v počátcích tak, aby byl bezpečný. Nikdo nepředpokládal, že se bude používat k předávání citlivých údajů. Dnes je bezpečnost clipboardu ve většině OS stále poměrně zanedbaná, pravděpodobně z toho důvodu, aby se zachovala jednoduchost používání.

Většina bezpečnostních hrozeb spojených s clipboardem napříč operačními systémy by se dala shrnout do následujících kategorií:

1. vložení škodlivých dat do clipboardu,
2. únik dat z clipboardu.

Zranitelnosti z těchto dvou kategorií jsou samozřejmě závislé na implementaci clipboardu. Vzhledem k tomu, že nějaká jejich podmnožina se týká prakticky každého operačního systému, budeme je v této kapitole popisovat z pohledu obecné implementace clipboardu.

2.1.1 Pastejacking

Pastejacking je typ útoku, který patří do kategorie zkopírování škodlivých dat do clipboardu. Je to útok, který funguje prakticky na všech konvenčních OS. V rámci této práce byl otestován konkrétně na macOS, Android, Windows a Linux. Jediná platforma, která nebyla testována, byl iOS. Vzhledem k tomu, že tato zranitelnost nesouvisí přímo s implementací clipboardu, je pravděpodobné, že by situace na iOS byla stejná jako na ostatních OS. [8]

Tento útok využívá clipboard API v jazyce JavaScript takovým způsobem, aby změnil obsah na škodlivý. Vzhledem k tomu, že interpreter jazyka JavaScript je zahrnut v každém webovém prohlížeči, je možné umístit tento útok na libovolném webu. Bezpečnostní opatření clipboardového API v jazyce JavaScript, zejména v souvislosti kopírování obsahu do clipboardu, nejsou dostatečná. Opatření jsou následující:

- musíme aktivně prohlížet obsah webu, čili musí být v popředí,

- kopírování v JavaScriptu musí být navázáno na nějakou událost, která pochází od uživatele (například zmáčknutí tlačítka nebo vyvolání události kopírování pomocí `Ctrl+C`...).
  - Bohužel u desktopových systémů s Chromium-based prohlížeči je akceptovatelná i událost načtení stránky, u mobilních ne.

Tato opatření sice zamezí tomu, že do clipboardu nemůže být nic vloženo, pokud uživatel explicitně nevykoná nějakou aktivitu. Nicméně obsah, který je reálně zkopírován, může být odlišný od obsahu, který uživatel označil myší a tedy chtěl zkopírovat.

Příklad zneužití pastejackingu popisuje tento scénář:

1. Uživatel navštíví webovou stránku, která se tváří jako manuálová stránka s příkazy pro Shell.
2. Uživatel bude chtít nějaký z příkazů použít, a proto ho zkopíruje.
3. Uživatel otevře terminál a jedním z možných způsobů vyvolá akci vložení obsahu ze schránky.
4. Po vyvolání akce se vloží škodlivý příkaz, který dělá něco naprosto odlišného.
5. Příkaz je spuštěn:
   a. pouze následkem akce vložení,
   b. samotným uživatelem díky neznalosti možných rizik.

Nyní si ukážeme kód, jakým bychom mohli dojít k naplnění tohoto scénáře:

Výpis kódu 2.1 Pastejacking cílený na cmd.exe. [9]

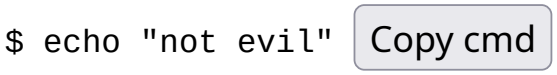
```
<!DOCTYPE html>
<html>
<body>
<code class="language-bash hljs">
$ echo "not evil"
</code>
<button onclick="myFunction()">Copy cmd</button>

<script>
document.addEventListener('copy', function(e){
    e.clipboardData.setData('text/plain', 'echo "evil"\r');
    e.preventDefault();
    });

function myFunction()
{
    navigator.clipboard.writeText('echo "evil"\r');
}
</script>

</body>
</html>
```

Výsledek webové stránky bude vypadat takto:



Obrázek 2.1 Webová stránka obsahující pastejacking.

Ve zdrojovém kódu je ukázka použití jak nového, tak staršího clipboardového API v jazyce JavaScript. Pomocí metody `addEventListener()` je nastavena jako callback funkce `function()`, která reaguje na událost kopírování. Funkce zamezí šíření původní události a nastaví obsah clipboardu na škodlivá data.

Druhá varianta je, že uživatel stiskne tlačítko na kopírování příkazu, v reakci na zmáčknutí je zavolána funkce, která nastaví obsah clipboardu na škodlivá data.

Tento scénář se týká především desktopových OS. Síla tohoto útoku je v tom, že naplnění clipboardu škodlivými daty se stane přesně v ten okamžik, kdy uživatel sám vyvolá akci kopírování. Tím pádem je poměrně pravděpodobné, že bude chtít obsah obratem vložit. Pokud se navíc jedná o web vypadající jako wiki nebo manuálová stránka, uživatel zřejmě skutečně vloží obsah do příkazového řádku. Zásadní moment z hlediska dopadu hrozby se odehrává v bodě 4 našeho scénáře. Pokud je možné v systému dosáhnout bodu 5a našeho scénáře, jedná se o poměrně vážnou hrozbu.

Z testovaných platforem v rámci této práce vyšel nejhůře OS Windows, konkrétně se jednalo o Windows 11. Jeho cmd.exe i PowerShell používají ke spuštění příkazu ukončovací znak `\r`, který je při běžné situaci vyvolán uživatelem pomocí stisknutí klávesy Enter. Nicméně oba dva terminály nezohledňují, zda znak přijde z klávesnice nebo z clipboardu, a tudíž akci spojenou s tímto řídícím znakem vyvolají. Pokud jsou tedy nastavena data clipboardu na `echo "evil"\r`, je akcí vložení spuštěn příkaz `echo "evil"`. Bylo by tedy možné místo neškodného demostračního příkazu použít takový, který způsobí buď škodu uživateli, nebo ukradne uživatelská data a pošle je na server útočníka. Pokud by uživatel vložil příkaz do konzole s administrátorskými právy, budou možnosti útočníka ještě širší a potencionální škody větší.

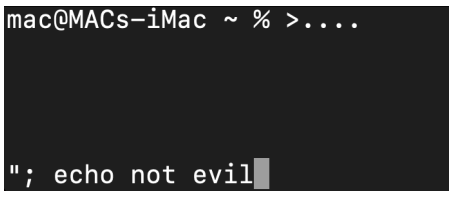
MacOS, konkrétně macOS Big Sur, vyšel z tohoto testu lépe. Data clipboardu lze změnit za jiné, než uživatel kopíroval, nicméně příchod znaku `\r` z clipboardu je interpretován jako text a nikoliv jako zmáčknutí klávesy Enter. Testována byl i znak `\n` se stejným výsledkem. Díky tomu se vložený obsah do terminálu nespustí pouze jeho vložením. Stejně dopadl i OS Linux u většiny distribucí.

Bohužel Linux Mint je jednou z výjimek. U této distribuce je situace stejná jako na Windows. Ochrana, která se používá u OS Linux k zamezení spuštění příkazu vložením, je spojená s funkcí `readline()`. Tato funkce se používá pro čtení znaků z terminálu. Chování této funkce se dá ovlivnit v konfiguračním souboru `~/.inputrc`. Pokud přidáme řádek `set enable-bracketed-paste on`, znamená to, že `readline()` umožní vložit každou vloženou položku do editační vyrovnávací paměti jako jeden řetězec znaků, místo aby s každým znakem zacházel, jako by byl přečten z klávesnice. To může zabránit tomu, aby vložené znaky byly interpretovány jako editační příkazy. Bohužel, tato ochrana není v Linux Mint aktivní. [10]

Útok může být ovšem navržen tak, aby se dospělo do bodu 5b, například pokud budou škodlivá data následující:

```
"echo 'evil'; \n\n\n\n\n.......\n\n; echo "not evil"
```

Výstup macOS terminálu bude po vložení vypadat následovně:



Obrázek 2.2 Terminál macOS po vložení obsahu clipboardu.

Pomocí vložení mnoha znaků `\n` docílíme vyčerpání vyrovnávací paměti terminálu a ten pak u tohoto systému reaguje jako na obrázku 2.2. Podobný efekt se odehraje i u terminálů OS Linux. Uživatel tedy vidí pouze příkaz, který kopíroval, a nějaké volné místo. Přinejhorším mu zmizí

prompt jeho shellu. Proto nemá podezření na nějaký škodlivý kód a potvrdí provedení příkazu. Tím dojde ke spuštění škodlivého kódu.

Další možností pastejackingu je necílit přímo na spuštění škodlivého kódu, ale cílit na navštívení škodlivého webu. Uživatel zkopíruje URL nějaké neškodné webové stránky, která je mu nabízena. Nicméně reálně je mu do clipboaru zkopírováno URL škodlivé stránky. Uživatel to nemusí registrovat a stránku navštíví. Toto může být již brána k efektivnějšímu phishingu nebo čemukoliv jinému, jako například nainstalování malwaru na uživatelův OS za pomoci klamavé reklamy, tedy spuštění škodlivého kódu až v druhém kroku. Pokročilejší clipboard managery navíc umějí navázat nějakou činnost v závislosti na kopírovaném obsahu, například automatické otevírání URL. Tím by se dosáhlo toho, že uživatel bude rovnou přesměrován na škodlivou stránku, aniž by zaregistroval změnu kopírovaného URL. Tyto možnosti jsou naštěstí v defaultním nastavení vypnuté napříč spektrem OS.

Následující tabulka popisuje vážnosti hrozeb pomocí metodiky CVSS:

Tabulka 2.1 Vyhodnocení hrozby pastejackingu.

Operační systém	Závažnost pastejackingu			
	Uživatelský terminál		Administrátorský terminál	
	Základní skóre	Závažnost	Základní skóre	Závažnost
Windows	7,1	high	9,6	critical
macOS	7,1	high	9,6	critical
Linux	7,1	high	9,6	critical

Podrobnější hodnocení této problematiky najdete v příloze B.1 a B.2.

2.1.2 Únik dat z clipboardu

Úniky dat z clipboardu jsou zásadní problematika. Vzhledem k tomu, že clipboard je velmi používaný, většina uživatelů dnes nepřemýšlí nad tím, zda je bezpečné ho na některá data používat. Vzhledem k tomu, že je clipboard navržen jako sdílené médium, mají k němu přístup de facto všechny aplikace. Nicméně při některých operacích copy–paste by bylo vhodnější chování spíše P2P. To znamená předat data ze zdrojové aplikace do cílové bez možnosti toho, aby někdo další mohl předávaný obsah vidět, obzvláště pokud se v clipboardu nachází například heslo. Únik dat z clipboardu bychom tedy mohli definovat jako vložení dat z clipboardu do nějaké aplikace bez toho, aby o tom uživatel věděl.

Z hlediska clipboardového API v jazyce JavaScript existují metody, které jsou schopné získat obsah clipboardu. Jedna z nich je `navigator.clipboard.readText()`. Její bezpečnost je zajištěna v každém webovém prohlížeči jiným způsobem:

Tabulka 2.2 Fungování funkce readText(). [9]

Firefox desktop a mobile	Funkce je podporována pouze v doplňcích prohlížeče, které mají navíc oprávnění pro manipulaci s clipboardem.
Chromium-based desktop	Funkce je plně podporována a při každém pokusu přistoupit ke schránce je uživatel dotázán, zda to chce umožnit. Nicméně událost, která je validní pro volání této funkce, je i načtení stránky.
Safari desktop a mobile	Funkce je plně podporovaná, ale je možné ji volat pouze s příchodem události `paste`.
Chromium-based mobile	Stejné chování jako u Safari.

V historii se dělo, že webové stránky mohly číst obsah clipboardu svých klientů a odesílat data k sobě na server. Díky tomu docházelo k únikům hesel, která byla prostřednictvím clipboardu předávána. Metody zabezpečení webových prohlížečů,které jsme si popsali v tabulce 2.2, pouze zmenšují vektor útoku tím způsobem, že zabrání webovým stránkám získávat obsah clipboardu. Nicméně stále je zde možnost, že libovolná lokální aplikace může získat obsah clipboardu bez vědomí uživatele, a pak například poslat clipboardová data zabalená v nějakém bug reportu prostřednictvím sítě na svůj server. Proto je snaha o nasazení mechanismů, které neumožní získat data z clipboardu bez uživatelova vědomí, a my si je nyní představíme.

1. Znemožnit přístup ke clipboardu aplikacím běžícím na pozadí (nejsou viditelné),
2. operaci kopírování/vložení smí udělat pouze okna, která jsou aktuálně zaměřená (uživatel na nich má kurzor a aktivně je používá),
3. informovat uživatele, jaká aplikace aktuálně vkládá data,
4. předávat data P2P.

Nejprve si rozebereme desktopové konvenční OS. MacOS Big Sur, který byl v rámci této práce analyzován, obsahuje nástroje jako je `pbcopy` a `pbpaste`. Tyto nástroje umožňují kopírovat, respektive vkládat obsah do, respektive z clipboardu pomocí přesměrování standardního vstupu, respektive výstupu. Je možné je zahrnout jako součást shellových skriptů, které běží na pozadí. Po podrobném prohlížení logů nebyly nalezeny žádné logy, které by monitorovaly komunikaci s clipboardem. Je tedy důvodné se domnívat, že žádný mechanismus od bodu 1 do bodu 3 není zde implementován. Vzhledem k tomu, že doba vydání verze Big Sur je stejná, jako doba vydání Apple iOS verze 14, který již řadu těchto bezpečnostních opatření nasadil, je důvodné se domnívat, že bezpečnost clipboardu macOS není pro Apple tak velkou prioritou. Taktéž nebyla zjištěna žádná informace o tom, že v nejnovější verzi macOS by se ohledně clipboardu udály nějaké změny. Z oficiální dokumentace [11][12] vyplývá, že se jedná o klasický centralizovaný přístup, kde pomocí globálního rozhraní jsou data umístěná fyzicky do clipboardu při operaci kopírování a při operaci vložení jsou z něj přečtena. Při novém kopírování je starý obsah nahrazen. Data je možné samozřejmě umisťovat ve více formátech. Zajímavé je, že macOS nemá svůj clipboard manager umožňující evidovat historii kopírování. [13]

Apple iOS je na tom s bezpečností clipboardu lépe. Ve verzi 14 je situace taková, že přístup ke clipboardu není umožněn aplikacím na pozadí, které nejsou aktivně používány. Vzhledem k tomu, že bylo zjištěno, že obsah clipboardu získávají i aktuálně používané aplikace bez uživatelova vědomí, byla implementována notifikace v podobě banneru, který uživatele při každém získávání dat z clipboardu informuje, která aplikace aktuálně vkládá data z clipboardu. Díky tomu uživatel ví přesně, kdo ke clipboardu přistupuje v daný moment. To ovšem nezabrání uniknutí dat. Proto byla v iOS 15 implementována funkce secure paste, která má zajistit, že k obsahu clipboardu není možné přistoupit, dokud uživatel explicitně nevykoná akci vložení. Vzhledem k tomu, že je tento OS uzavřený, nejde explicitně říct, že je tato ochrana tak silná, jak Apple uvádí. Pokud by to tak bylo, splňoval by iOS 15 všechny body od 1 až 4 s tím, že nemůžeme přesně říci, jestli P2P znamená, že tok dat jde přímo ze zdrojové aplikace do cílové, nebo zda se jedná o klasický centralizovaný model clipboardu pouze s přísnějšími oprávněními. Podle clipboardového API tohoto systému je možné k předávání dat prostřednictvím clipboardu použít Content Provider, který umožňuje P2P přenášení dat. Ovšem bezpečnost tohoto způsobu předávání dat je odvozena od implementace Content Provideru daného klienta. Podrobnější informace by vyžadovaly podrobnější bezpečnostní analýzu, které se tato práce nevěnuje u tohoto konrkétního OS. [4]

Windows, konkrétně Windows 11, je na tom podobně jako macOS. Obsahuje nástroje, které umožňují přistupovat ke clipboardu v shellovém scriptu bez nutnosti existence okna. V PowerShellu existují nástroje `Get-Clipboard` a `Set-Clipboard`, které jsou ekvivalenty nástrojů `pbcopy` a `pbpaste` v macOS. Podle popisu clipboard API a obecně clipboardu v oficiální dokumentaci [3] se opět jedná o model centralizovaného clipboardu. Žádné logování nebo bannery oznamující

informace o aktivitě clipboardu zde také nejsou. Znamená to tedy, že žádný z námi popsaných principů 1 až 4 není uplatněn. Navíc je zde možné zapnout evidování historie clipboardu a tedy vkládat obsah, který již byl nahrazen novým. Tím pádem je vektor možných úniků větší, neboť oficiální dokumentace nabízí API, pomocí kterého je možné získat jednotlivé položky historie clipboardu. Navíc, pokud se vyčistí clipboard konvenčním způsobem, je smazán pouze aktuální výběr, ale v historii clipboardu obsah zůstane, pokud není explicitně zavolána funkce API pro vyčištění historie clipboardu. [6][3][14][15]

Android a desktopové operační systémy Linuxového typu si probereme v kapitole 3, která se věnuje podrobnější analýze Linuxových implementací. Nyní si pouze zmíníme, že veškeré Linuxové implementace se snaží o P2P clipboard. Android od verze 12 používá bannery stejným způsobem jako Apple iOS ve verzi 14, nicméně bannery je možné vypnout. V Androidu a desktopových OS Linux často existuje evidence historie clipboardu, která P2P přístup posléze degraduje na centralizovaný. O clipboard managerech si také povíme více v sekci 3. V desktopových OS Linux s protředím KDE Plasma je jako výchozí clipboard manager používán Klipper. Klipper poskytuje API, které umožňuje získávat z něj uloženou historii clipboardu. [7][16]

2.2 Problematika předávání hesel

Hovořili jsme již o tom, že clipboard může obsahovat citlivá data. Jednou z množin velice zásadních dat, která by neměla uniknou, jsou hesla. Vzhledem k tomu, že lidé mají tendenci používat slabá hesla a nebo pouze jedno heslo pro několik různých účtů různých webových služeb, byl vytvořen správce hesel, do kterého se uloží silná hesla zašifrovaná jedním bezpečným heslem, které si uživatel bude pamatovat.

Zásadní problém ovšem spočívá v předání tohoto hesla do cílové aplikace. Vzhledem k tomu, že se jedná o silná hesla, tedy hesla s vysokou entropií, obsahují často speciální symboly a hesla jsou často dlouhá. Proto přepsání takového hesla pomocí klávesnice by bylo pro uživatele značně nepohodlné, a navíc by heslo někdo mohl vyfotit při jeho přepisování uživatelem. Proto se jako první nejjednodušší řešení nabízí použít clipboard.

Použití clipboardu k předávání hesel by bylo dobré řešení, pokud by clipboard fungoval bezpečně. Vzhledem k tomu, že clipboard podporují prakticky všechny grafické aplikace, nabízí se pro předávání hesel jako logická volba. Pokud používáme správce hesel, který je implementovaný uvnitř námi používané aplikace (například ukládání hesel ve webovém prohlížeči), jsme omezeni pouze na tuto aplikaci. Pokud ale chceme správce hesel pro všechny lokální aplikace, musíme s nimi umět komunikovat. Proto je zapotřebí nějaký univerzální způsob, jakým heslo předat.

V podstatě většina populárních správců hesel podporuje předávání hesel pomocí clipboardu. Jmenovitě se jedná o Bitwarden, KeePass nebo v Linuxových systémech oblíbený Pass. Předávání probíhá tím způsobem, že po odemknutí správce hesel je heslo zkopírováno do clipboardu. Následně je vloženo do místa, kam je potřeba. Poté by mělo následovat vyčištění clipboardu. Problematiku toho přístupu bychom mohli rozdělit do dvou kategorií:

1. dostatečné vyčištění clipboardu,
2. velikost časového intervalu před vyčištěním clipboardu.

Způsob, jakým čistí obsah clipboardu Bitwarden, je příkladem problematiky bodu 1. Vzhledem k tomu, že při zkopírování nějakého nového výběru do clipboardu jsou stará data nahrazena vždy novými, je čistění clipboardu zpravidla provedeno tak, že se zkopíruje prázdný řetězec. Pokud je zde ovšem nějaký clipboard manager, který eviduje historii clipboardu, je heslo okamžitě evidováno jako jedna z položek historie clipboardu. Čili aktuální clipboard je prázdný, ale heslo je možné dohledat v historii clipboardu. Pokud explicitně nezavoláme nějakou funkci (případně metodu), kterou poskytuje API používaného clipboard manageru, heslo zde zůstane uložené tak dlouho, dokud nebude přepsáno novou položkou. V desktopových OS Linux je navíc ukládána historie clipboardu clipboard managery do souborů na pevném disku. Například Klipper ukládá

svou historii do `~/.local/share/klipper/history2.lst`, což znamená, že heslo zůstane uložené v otevřeném textu i po vypnutí počítače. Únik hesla je pak zjednodušen na monitorování tohoto souboru.

Clipboard manager umožňuje evidovat různé množství položek historie. U OS Windows je velikost historie clipboardu standardně 25 položek, u Klipperu 20, nicméně je možné tuto kapacitu navýšit až na 2048 položek. Pokud by se zde tedy ocitlo heslo, muselo by se zkopírovat tolik různých výběrů, abychom přesáhli kapacitu clipboard manageru, což může trvat poměrně dlouhou dobu. [7]

Správci hesel, pro které není tak klíčové být multiplatformní, umějí zavolat explicitně vyčištění historie clipboardu. Jedním z nich je například Pass. Všimněme si, jakým způsobem celá operace kopírování hesla včetně jeho vyčištění, funguje:

Výpis kódu 2.2 Funkce na kopírování hesel programu Pass. [17]

```
clip() {
        if [[ -n $WAYLAND_DISPLAY ]] && command -v wl-copy &> /dev/null;
        then
                local copy_cmd=( wl-copy )
                local paste_cmd=( wl-paste -n )
                if [[ $X_SELECTION == primary ]];
                then
                        copy_cmd+=( --primary )
                        paste_cmd+=( --primary )
                fi
                local display_name="$WAYLAND_DISPLAY"
        elif [[ -n $DISPLAY ]] && command -v xclip &> /dev/null;
        then
                local copy_cmd=( xclip -selection "$X_SELECTION" )
                local paste_cmd=( xclip -o -selection "$X_SELECTION" )
                local display_name="$DISPLAY"
        else
                die "Error: No X11 or Wayland display and clipper detected"
        fi
        local sleep_argv0="password store sleep on display $display_name"

        # This base64 business is because bash cannot
        # store binary data in a shell
        # variable. Specifically, it cannot store
        # nulls nor (non-trivally) store
        # trailing new lines.
        pkill -f "^$sleep_argv0" 2>/dev/null && sleep 0.5
        local before="$("${paste_cmd[@]}" 2>/dev/null | $BASE64)"
        echo -n "$1" | "${copy_cmd[@]}" || die \
                  "Error: Could not copy data to the clipboard"
        (
                ( exec -a "$sleep_argv0" bash <<<"trap 'kill %1' TERM; \
                  sleep '$CLIP_TIME' & wait" )
                local now="$("${paste_cmd[@]}" | $BASE64)"
                [[ $now != $(echo -n "$1" | $BASE64) ]] && before="$now"
                # It might be nice to programatically check
                # to see if klipper exists,
                # as well as checking for other common
                # clipboard managers. But for now,
                # this works fine -- if qdbus isn't
                # there or if klipper isn't running,
                # this essentially becomes a no-op.
                # Clipboard managers frequently write
```

```
                #their history out in plaintext,
                # so we axe it here:
                qdbus org.kde.klipper /klipper \
                  org.kde.klipper.klipper.clearClipboardHistory &>/dev/null

                echo "$before" | $BASE64 -d | "${copy_cmd[@]}"
        ) >/dev/null 2>&1 & disown
        echo "Copied $2 to clipboard. Will clear in $CLIP_TIME seconds."
}
```

Nejprve dojde ke zjištění, zda se jedná o Wayland prostředí nebo X.Org, a v závislosti na tom je zvolen buď nástroj xclip nebo wl-copy/wl-paste. Tyto nástroje slouží k manipulaci s clipboardem z příkazového řádku. Obsah clipboardu je vložen pomocí tohoto nástroje do proměnné. Posléze je do clipboardu umístěno heslo. Po uplynutí časového intervalu je čištěný provedeno pomocí tohoto příkazu:

```
qdbus org.kde.klipper /klipper \
   org.kde.klipper.klipper.clearClipboardHistory &>/dev/null
```

Program `qdbus` umožňuje komunikaci s Qt aplikacemi prostřednictvím komunikačního rozhraní D-Bus. Pomocí těchto parametrů pošle Klipperu příkaz k tomu, aby vymazal veškerou historii clipboardu. Následně je zkopírován do clipboardu původní obsah, který obsahoval před předáním hesla.

Pokud není používán žádný clipboard manager, heslo je smazáno tím, že je zkopírován do clipboardu obsah, který se v něm nacházel před heslem. Problém ovšem nastává, pokud je používán jiný clipboard manager než Klipper. Pass s tím nepočítá, a tím pádem zůstane heslo zaevidováno v historii clipboardu v otevřeném textu.

Nyní proberme problematiku časového intervalu, tedy bodu 2. Pokud budeme používat KeePass na OS Windows a budeme používat clipboard k předání hesla, KeePass umí zajistit, aby heslo nebylo evidováno pomocí clipboard manageru jako jedna z položek historie clipboardu. Čeká 20 sekund na to, než uživatel heslo zkopíruje, a následně vyčistí clipboard. To je asi zatím nejlepší možné řešení předávání hesla pomocí clipboardu, které jsme si doposud představili. Problém je ovšem v tom, že než heslo stačíme zkopírovat a než je odstraněno z clipboardu, může zde být aplikace, která jednoduše toto heslo přečte,a bohužel tomu není zabráněno ani takto dobrou konfigurací a řešením, jaké nabízí KeePass.

V podstatě na libovolném desktopovém OS nebudeme vědět, kdo si heslo zkopíroval, jestli bylo zkopírováno i do jiné aplikace než do námi požadované... Pokud bude situace tak dobrá, že uživatel bude skutečně vědět, kdo s clipboardem manipuluje, a budou tedy aplikována opatření, která jsme si popsali v sekci 2.1.2, úniku hesla v podstatě zabráněno stále není. Pouze je pomocí krátkého časového intervalu a ostatních bezpečnostních opatření snížena pravděpodobnost úniku, případně pomocí informačních bannerů ohledně vkládání z clipboardu je možné identifikovat škodlivou aplikaci dříve, než dojde k úniku citlivých dat.

Z tohoto důvodu byly vytvořeny jiné mechanismy na předávání hesel, jako je automatické vyplnění hesla. Taková funkcionalita se týká především mobilních platforem, které mají pro tento účel vytvořeno systémové API. KeePass umožňuje například použít metodu auto-type, což posílá události zmáčknutí kláves odpovídající jednotlivým znakům hesla do cílové aplikace, takže je heslo de facto napsáno virtuální klávesnicí. Tato řešení jsou často závislá na používané platformě a pro koncového uživatele je clipboard pohodlnější variantou, kterou si může vybrat. [18]

Spolehlivé zabránění úniků hesel/citlivých dat z clipboardu je jedním z hlavních cílů této práce. Rozbor možností, jak toho dosáhnout, a implementace jedné z nich je popsána v kapitole 5.

Kapitola 3

Linuxový clipboard

Tato sekce podrobně popisuje fungování clipboardu v OS Linuxového typu. Jsou zde představeny veškeré aktuálně používané implementace clipboardu a jsou analyzovány jejich specifické bezpečnostní aspekty.

3.1 Problematika P2P clipboardu

Linuxové clipboardy jsou zpravidla P2P clipboardy. To znamená, že sdílený prostor obsahuje pouze metadata a jeho účelem je dohodnout parametry přenosu dat. Přenos dat jako takových probíhá P2P z kopírující aplikace do vkládající na základě dohodnutých parametrů. Problematika tohoto konceptu je v tom, že pokud se něco zkopíruje a aplikace poskytující data se zavře, zaniknou s ní veškeré objekty, což způsobí, že clipboard bude prázdný. Aby se toto chování nedělo, jsou často používány clipboard managery. Ty, kromě evidování historie clipboardu, mají funkci potlačit tento jev. Při zavření aplikace, která byla poskytovatelem dat, se stane to, že clipboard manager se uchází o to, aby byl aplikací, která poskytuje data.

Clipboard manager musí nejprve data od aplikace získat, aby je mohl poskytovat a evidovat v rámci historie. Toho docílí tak, že při změnách obsahu clipboardu má roli aplikace, která chce data vložit (tím myšleno přijmout). Data jsou mu tím pádem poskytnuta a předání dat proběhlo mezi kopírující aplikací a clipboard managerem P2P. To ovšem dělá z P2P clipboardu do značné míry centralizovaný clipboard, protože data jsou fyzicky obsažen v clipboard manageru.

3.2 Primary selection

Primary selection je často používaný nástroj v desktopových Linuxových OS. Jedná se de facto o druhý clipboard. Již jsme si naznačili v sekci 1.2, jakým způsobem může druhý clipboard fungovat. Funguje tedy tak, že se označí text pomocí kurzotu myši. Následně se klikne na místo, kam chceme text vložit. Zmáčknutím prostředního tlačítka je obsah vložen. [19][1]

Bezpečnostní problém je ovšem v tom, že pouhým označením nějakého textu myší je již text zkopírovaný, a je možné ho vložit. Operace označení textu myší ovšem neznamená, že máme zájem data kopírovat. Můžeme například chtít vymazat nějaký text hromadně. Pokud editujeme nějaká citlivá data, nechceme, aby ostatní aplikace měly možnost tato data vložit bez našeho vědomí. Nicméně tuto možnost mají, protože data jsou fakticky zkopírována bez našeho vědomí tím, že jsme je označili.

V konkurenčním macOS můžeme vidět, že tuto utilitu je možné používat pouze v rámci terminálu, tedy na úrovni samostatné aplikace.

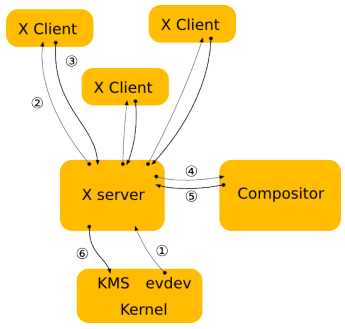
3.3 MIME type

MIME type, neboli typ média, je identifikátor složený ze dvou částí, který definuje formát a obsah souboru/dat. Původně byl navržen pro data přenášená internetem. Tento standard byl poprvé definován v RFC 2045. Nyní je tento standard již hodně bohatý, a proto je specifikovaný v několika RFC a je na něj dohlíženo organizací Internet Assigned Numbers Authority. Zkratka MIME znamená Multipurpose Internet Mail Extensions. Vzhledem k tomu, že se standard začal používat v širším měřítku než pouze v emailové komunikaci, používá se dnes souslovím MIME type především ve významu media type. V tomto kontextu je zkratka také používaná v této práci. [20][21]

MIME typy se používají v komunikaci s clipboardem v OS Linuxového typu. Výhody a důvody posílání metadat definujících formát/obsah předávaných dat již byly popsány v sekci 1.3.

3.4 X.Org clipboard

X.Org je open source implementace X Window System, která je stále nejpoužívanější v OS Linuxového typu. Slouží k vykreslování oken, ale také umožňuje předávání dat mezi okny. Architektura tohoto systému vypadá takto:



Obrázek 3.1 Architektura X.Org. [5]

Základní API umožňující komunikaci s X serverem poskytuje knihovna Xlib. V dnešní době existuje již novější knihovna s názvem XCB, nicméně Xlib je plně podporovaná. Komunikace mezi klienty a X serverem zpravidla probíhá přes UNIX domain sockets, nicméně může pobíhat také přes TCP. Dokonce je možná i kombinace. Jeden klient může komunikovat prostřednictvím TCP a druhý prostřednictvím UNIX domain sockets. [19][1]

Funkčnost clipboardu je zde implementována přes takzvané výběry. Ty jsou identifikovány pomocí unikátních identifikátorů, atomů. Atom je v dvousložkový identifikátor, jednou složkou je název v podobě řetězce a druhou složkou je identifikační číslo atomu v podobě celočíselného 32 bitového typu. [19][1]

Výběry jsou používané ve spojitosti s clipboardem dva. Jeden se jmenuje PRIMARY a druhý CLIPBOARD. PRIMARY reprezentuje primary selection, který jsme si popsali v sekci 3.2. CLIPBOARD se používá pro klasické kopírování/vkládání pomocí `Ctrl+C`/`Ctrl+V`. Tyto metody se nijak neliší z hlediska mechanismu předávání dat nebo implementace. Jediný rozdíl je v pojmenovaní, jehož účelem je docílení dvou segregovaných clipboardů. [19][1]

Pro předávání dat je nutná existence okna. Data jsou totiž předávána pomocí takzvaných vlastností oken. Ty jsou identifikovány opět pomocí atomů. Klienti tedy můžou ke svým oknům připojit několik vlastností. Hodnota těchto vlastností je pole osmi, šestnácti nebo třiceti dvou bitových hodnot, jejichž typy odpovídají `char`, `short` a `long`. [19][1]

Klient, který chce být poskytovatelem dat, musí nejprve získat vlastnictví výběru. Toho docílí následujícími dvěma kroky. Na základě jména výběru, se kterým chce pracovat, získá

pomocí funkce `XInterAtom()` od serveru číselný identifikátor atomu. Následně zavolá funkci `XSetSelectionOwner()`, která vygeneruje pro server žádost o přidělení vlastnictví výběru, který je specifikován dodaným atomem. V případě, že se klient stane vlastníkem výběru, musí kontrolovat smyčku událostí. [19][1]

Přijde-li od serveru událost typu `SelectionClear`, někdo jiný se stal vlastníkem výběru. Pokud ale přijde událost typu `SelectionRequest`, žádá nás nějaký z klientů o data. Událost tohoto typu je struktura, která obsahuje identifikátor okna protistrany, atom identifikující vlastnost tohoto okna, do které chce protistrana zapsat data, atom identifikující název formátu, ve kterém mají být data zapsána a další informace, které pro nás nejsou až tak zásadní k pochopení problematiky. [19][1]

Data jsou zapsána do vlastnosti okna protistrany pomocí funkce `XChangeProperty()`. Následně se sestrojí událost typu `SelectionNotify` obsahující strukturu typu `XSelectionEvent`. Ta se odešle serveru pomocí funkce `XSendEvent()` a ten ji předá cílovému klientovy. Událost mu oznamuje, že požadovaná data jsou již zapsána v požadovaném typu a vlastnosti jeho okna. [19][1]

Klient, který chce data vložit, postupuje následovně. Ověří, zda má výběr vlastníka pomocí funkce `GetSelectionOwner()`, které předá v parametru atom výběru. Pokračuje vytvořením okna a nového atomu pomocí funkce `XInternAtom()`, jež bude posléze identifikovat vlastnost jeho okna. Zbývá mu získat atom definující typ, ve kterém data požaduje. Poté má již vše potřebné k zavolání funkce `XConvertSelection()`, do jejíž parametrů zahrne získané, což je identifikátor okna, atom vlastnosti okna, atom výběru, atom definující typ dat a další parametry. Funkce vygeneruje žádost pro server. Ten posléze vytvoří událost `XSelectionRequestEvent`, kterou zašle protistraně. Následně klient kontroluje smyčku událostí a čeká na příchod události typu `SelectionNotify`, která oznamuje, že data byla přijata. Data z vlastnosti je možné získat pomocí funkce `XGetWindowProperty()`. Následně by měla být vlastnost zničena zavoláním funkce `XDeleteProperty()`. [19][1]

Aby bylo možné dohodnout formát dat, používá se často v první žádosti o data formát TARGETS, na který by měl klient poskytující data zareagovat tak, že pošle protistraně všechny dostupné formáty, ve kterých může data předat. Tyto formáty už jsou typu MIME. Další krok je již takový, jak jsme si ho popsali ve zjednodušeném modelu předávání dat, že klient požadující data posílá žádost o data již s formátem, ve kterém mu budou data poskytnuta protistranou. [19][1]

Nejedná se tedy o klasickou centralizovanou implementaci clipboardu, která by umístila data fyzicky do sdílené paměti. Data jsou umístěna do vlastnosti okna cílového klienta. Nicméně předání dat probíhá přes X server. Vlastnosti oken jsou ukládány na X serveru a alokovány v jeho paměti. Proto se jedná o „falešný“ P2P clipboard, tím je myšleno P2P clipboard s větší mírou centralizace. V sekci 3.5 uvidíme lepší koncept P2P clipboardu. [19][1]

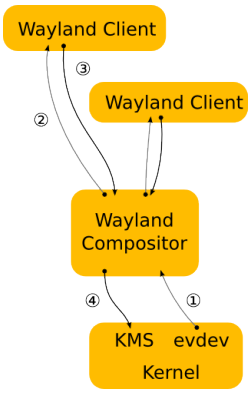
3.4.1 Fungování clipboard manageru

Clipboard manager je implementovaný tak, že neustále kontroluje vlastnictví výběru. Pokud výběr nemá žádného vlastníka, okamžitě se přihlásí o vlastnictví. Po jeho získání může poskytovat různá data z historie clipboardu ke kopírování. Primárně samozřejmě poskytuje poslední kopírovaná data. V případě, že vlastnictví převezme někdo jiný, je jednou z prvních aplikací, která okamžitě posílá žádost o data clipboardu, která zpravidla posléze obdrží. Díky tomu se nestává, že by následkem ukončení aplikace poskytující data již nebylo možné tato data vložit. Klient poskytující data sice zanikne, ale jeho roli převezme clipboard manager, který se stává novým poskytovatelem dat.

3.5 Wayland clipboard

Wayland je komunikační protokol, který specifikuje komunikaci mezi zobrazovacím serverem a jeho klienty. Jedná se o nástupce X.Org. Zobrazovací server využívající tento protokol se nazývá Wayland kompozitor. [5]

Hlavní funkce Waylandu je umožnit vykreslování oken a vše, co je s tím spojené, včetně předávání dat mezi jednotlivými grafickými aplikacemi. To, co Wayland pevně definuje, je komunikační protokol mezi Wayland kompozitorem (hraje roli zobrazovacího serveru) a klienty. Implementace kompozitoru je většinou zakomponována do projektů, které vyvíjí grafická pracovní prostředí. Například prostředí KDE Plasma nabízí svou implementaci s názvem KWin, prostředí GNOME nabízí Mutter. V rámci projektu Wayland existuje referenční implementace kompozitoru s názvem Weston, kterou jsou pak specifické kompozitory inspirovány. Architektura Waylandu vypadá takto:



Obrázek 3.2 Architektura Waylandu. [5]

Protokol Wayland je asynchronní objektově orientovaný protokol. Je napsaný v jazyce C. To znamená, že když budeme mluvit o objektech spojených s Wayland protokolem, fakticky se nejedná o instance nějakých třídy, tím myšleno typu class, ale jedná se o instance typu struct. Všechny žádosti jsou vyvoláním metody na nějakém objektu. Žádosti zahrnují ID objektu, které jednoznačně identifikuje objekt na serveru. Každý objekt implementuje nějaké rozhraní. Žádosti zahrnují operační kód, který určuje, jaká metoda tohoto rozhraní bude vyvolána. [5]

Každá událost je vytvořena a odeslána z objektu. Událost obsahuje ID objektu a operační kód, ze kterého může klient určit typ události. Události jsou generovány jak v reakci na žádosti, tak samostatně, když se změní stav serveru. Klienti naslouchají událostem a ukládají je do svých mezipamětí. Není potřeba se dotazovat serveru na změny jeho stavu. [5]

K rozhraním (globálním objektům) serveru (kompozitoru) je možné se připojit. Událostem přicházejícím z námi připojených rozhraních naslouchají listenery. Listenerům se nastaví adresy funkcí, které bude listener volat v závislosti na typu příchozí události. Volané funkce jsou tedy callbacky reagující na příchod události. Tímto způsobem funguje i Waylandový clipboard. [5]

3.5.1 Protokoly Wl_data a Zwp_primary_selection

Tyto dva protokoly slouží ke komunikaci s clipboardem. Protokol Wl_data je rozšířenější a je přímo od tvůrců Waylandu. Protokol zwp_primary_selection je externí protokol, který je podporován kompozitory jako je Mutter a KWin, Weston jej nepodporuje. Tento externí protokol je waylandovou formou primary selection. My budeme popisovat a analyzovat Wl_data protokol, konkrétně tu část protokolu, která pracuje s clipboardem (Wl_data také pracuje s mechanismem drag-and-drop). Externí protokol zwp_primary_selection je stejný, jako část protokolu

Wl_data, kterou analyzujeme. Jeho používání se liší od používání Wl_data pouze v jiném prefixu jeho názvu. V kompozitorech jsou oba protokoly implementovány stejně, ale každý funguje na jiné instanci clipboardu, čímž je dosažena existence dvou izolovaných clipboardů. [22][5]

Rozhraní clipboardu je reprezentováno objektem `wl_data_device_manager`. Ten reaguje na následující žádosti:

- `create_data_source()`,
  - Vytvoří na serveru objekt, který bude posléze použit klientem pro představení dat, která chce poskytnout protistraně, a spojí ho s ID dodaným v této žádosti.
  - Objekt reprezentující rozhraní na straně klienta namapované na vzdálený objekt serveru se jmenuje `wl_data_source`.
- `get_data_device()`.
  - Vytvoří na straně klienta objekt `wl_data_device` asociovaný s aktuálním sezením klienta a namapuje opět ID z klientovy žádosti na vzdálený objekt serveru. Sezení reprezentuje skupinu vstupních zařízení. Tento objekt souží k řízení clipboardu a zprostředkovává rozhraní pro přijímání dat clipboardu.

Po tom, co klient vytvoří tyto dva objekty, je musí svázat s listenery a na jednotlivých listenerech musí být již nastaveny funkce, které budou volány s příchodem událostí. Událostí, které mohou přijít ve spojitosti s objektem `wl_data_source`, je několik, nicméně nás budou zajímat hlavně událost `send()` a `cancelled()`. Ostatní se týkají mechanismu drag-and-drop, nicméně v listeneru nemůžeme jednoduše místo adresy na nějakou funkci nechat NULL, protože volání funkce na adrese NULL není přípustné. Proto postačí přidat prázdné funkce. Události, které budou přicházet ve spojitosti s objektem `wl_data_device`, také není možné ignorovat. Jedna z událostí, která nesmí být klientem ignorována, je `data_offer()`. Ta totiž představuje nový objekt `wl_data_offer`, který používá strana přijímající data. Nicméně i strana, která data poskytuje, může být zároveň ta, jenž je přijímá (kopírujeme nějaká data v rámci jedné aplikace), proto je nutné, aby tento objekt uchovala a s příchodem nové události `data_offer()`, která představuje nový objekt `wl_data_offer`, vyslala žádost `destroy()` spojenou se starým objektem `wl_data_offer`. Tato žádost dává serveru pokyn ke zničení starého objektu. Jinak by byly alokovány stále nové prostředky, které by nebyly uvolňovány. [23][5]

Nyní je již možné přejít k akci kopírování. Ta začíná tím, že pomocí žádosti `offer()` spojené s objektem `wl_data_source` sdělí klient serveru, v jakých MIME typu je schopný data poskytnout protistraně. Pro každý MIME typ se volá jedna žádost `offer()`. Vzdálený objekt serveru reaguje na tuto žádost tak, že přidá typ, který je obsažen v žádosti, k ostatním typům, které tento objekt nabízí (jsou drženy ve struktuře typu `wl_array`). [23][5]

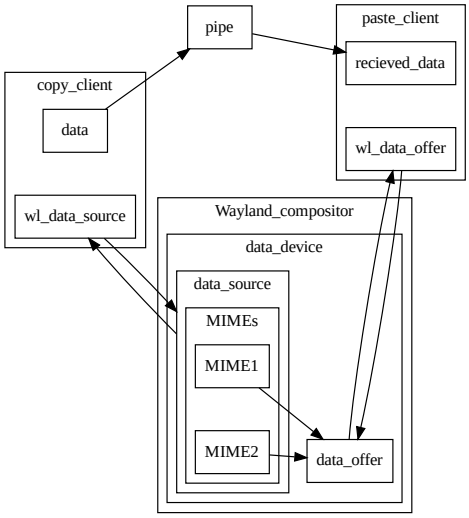
Posledník krokem je zaslání žádosti `set_selection()` spojené s objekty `wl_data_device` a `wl_data_source`. Tato žádost požaduje po serveru, aby byl serverový objekt, který je protějškem klientského objektu `wl_data_source`, nastaven jako aktuální obsah clipboardu. V žádosti `set_selection()` je kromě již zmíněných objektů (respektive jejich ID) také zakomponováno sériové číslo události, která akci kopírování vyvolala. Tato událost musí pocházet z nějakého vstupního zařízení (například klávesnice). [23][5]

Ve chvíli, kdy je naše nabídka nahrazena novější nabídkou (zkopírujeme opět něco jiného), je ze serveru odeslána událost `cancelled()` klientovi, který vytvořil nabídku předchozí. Tato událost je spojená s objektem `wl_data_source` a sděluje nám, že naše nabídka je již zastaralá a měli bychom ji zničit. Reaguje se tedy vyvoláním žádost `destroy()` spojené s `wl_data_source`, která dává serveru pokyn, aby zničil neaktuální data source objekt na serveru. [23][5]

Pokud má nějaký klient zájem o naše data a vyvolá operaci vložení, je nám serverem zaslána událost `send()`. Tato událost spojená s objektem `wl_data_source` obsahuje také MIME type, ve kterém chce protistrana data předat, a file descriptor, prostřednictvím kterého máme data zapsat. Tyto informace jsou nám předány v parametrech funkce, která je callback reagující na

tuto událost. Vzhledem k tomu, že celý protokol funguje přes UNIX domain sockets, které umí prostřednictvím dodatečné zprávy předat file descriptor, je toho zřejmě využito. File descriptor je předán tak, že cílové aplikaci (příjemce file descriptoru) otevře nový file descriptor, který odkazuje na stejnou datovou strukturu, jako odkazoval původní file descriptor v původním procesu. Proto přijetím žádosti `send()` již máme otevřený file descriptor odkazujícím na rouru, do které máme zapsat data pomocí volání jádra `write()`, a následně file descriptor uzavřít voláním `close()`. Druhý konec roury je otevřen v klientovi, který žádá o naše data. [23][5]

Všimněme si, že data nebyla předána nějakému prostředníkovi, pouze metadata. Jedná se tedy o plnohodnotný P2P clipboard, který si pouze přes sdílené médium dohodne způsob předání dat a jejich formát, ale samotná data se předávají P2P. Následující graf, který byl vytvořen pro tuto práci, zachycuje aktuální situaci předávání dat s podstatnými rozhraními pro pochopení problematiky:



Obrázek 3.3 Architektura Wayland clipboardu při předávání dat.

Zbývá nám popsat operaci vložení. Nejprve je klientem vytvořen objekt `wl_data_device` pomocí rozhraní `wl_data_device_manager` tak, jako tomu bylo v předchozím případě. Stejně tak je nutné svázat listener s objektem `wl_data_device`. V tuto chvíli jsou pro nás klíčovými událostmi `data_offer()` a `selection()`, které jsou spojené s objektem `wl_data_device`. Jak jsme si již řekli, událost `data_offer()` představuje nový objekt, který se používá pro operaci vložení. To znamená, že v parametru callbacku reagujícím na tuto událost je obsažen pointer na objekt `wl_data_offer`. V tomto callbacku je vykonat akci svázání listeneru s tímto objektem. Událost spojenou s objektem `wl_data_offer`, kterou budeme zpracovávat, je `offer()`. Tato událost nám v parametru callbacku, který na ní reaguje, předává dostupný MIME typ, ve kterém můžeme data požadovat. Je-li dostupných MIME typů více, přijde pro každý MIME typ nová událost tohoto typu. [23][5]

Posledním krokem je vyčkání na příchod události `selection()`, která je spojená s objektem `wl_data_device`. Callback, který na tuto událost reaguje, předává pomocí parametru opět pointer na objekt `wl_data_offer` (mělo by se jednat o stejný objekt, jako v události `data_offer()`). Nyní již stačí vytvořit rouru, kterou budou přijata data od protistrany, a následovně vyvolat žádost `receive()` spojenou s objektem `wl_data_offer`. Žádost musí krom objektu obsahovat MIME typ (mělo by se jednat o jeden z dostupných, který nám byl sdělem prostřednictvím události `offer()`), ve kterém data požadujeme, a file descriptor odkazující na zapisovací konec roury.

Po předání file descriptoru do žádosti musí být zavřen pomocí volání `close()` a z file descriptoru odkazujícího na čtecí konec roury by mělo být okamžitě čteno pomocí volání jádra `read()`. Data jsou po zapsání klienta, který poskytuje data, přečtena a proces vložení je dokončen. [23][5]

3.6 Android clipboard

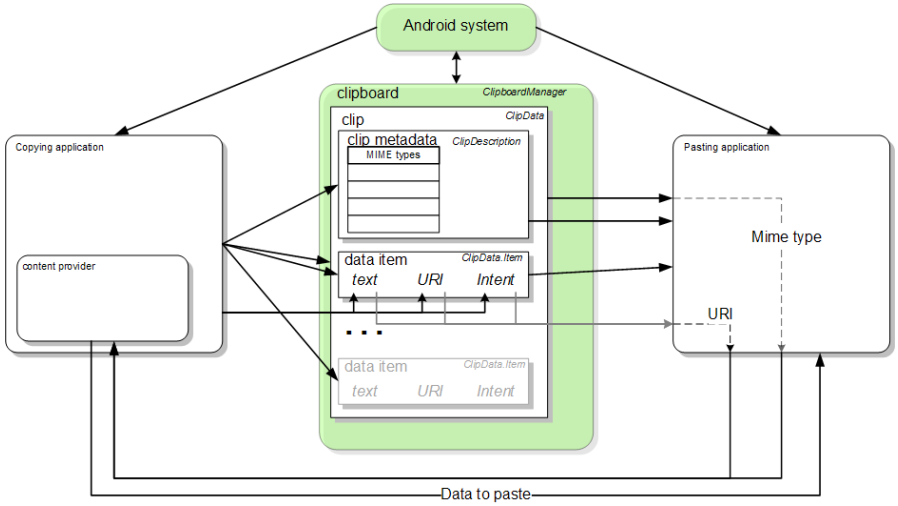
Operační systém Android nepoužívá ani jednu z konvenčních implementací clipboardu, které jsme si již popsali, ale nabízí svou vlastní implementaci clipboardu. Pro práci s clipboardem můžeme použít clipbordové API v jazyce Java, které si nyní popíšeme.

Clipboard je reprezentován globální třídou `ClipboardManager`. Instance této třídy se nevytváří, nýbrž se pomocí statické metody `getSystemService(CLIPBOARD_SERVICE)` získává reference na existující instanci. [16]

Kopírování obsahu se provádí metodou této třídy `setPrimaryClip()`, která umístí do clipboardu objekt typy `ClipData`. Pokud tento objekt již v clipboardu existuje, je nahrazen novým. Vložení obsahu z clipboardu se provádí metodou `getPrimaryClip()`. Tato metoda vrací obsah clipboardu v podobě objektu typu `ClipData`. [16]

Objekt `ClipData` obsahuje jeden objekt typu `ClipDescription`, který obsahuje MIME typy, ve kterých je možné data poskytnout. Dále objekt obsahuje jeden nebo více objektů typu `Item`. `Item` může obsahovat data jako je text, html text, URI identifikátor a také `Intern`. Tyto dva objekty se často nemusí vytvářet zvlášť a následně je předávat do nového objektu typu `ClipData`, neboť třída `ClipData` obsahuje statické metody, které vytvoří tento objekt již naplněný požadovaným obsahem (`newPlainText()`, `newUri()`, ...). Do této doby se jedná o centralizovaný model clipboardu a taky tímto způsobem funguje. Nicméně pro předávání komplexnějších datových struktur je možné zapsat obsah ve formě URI, které odkazuje na Content Provider. [16]

Content Provider je definován abstraktní třídou `ContentProvider`, jež musí být implementována v aplikaci poskytující data. Jedná se především o metody `query()`, pomocí které získává volající data, a `getType()`, která vrací MIME typ dat. Metoda `getType()` je volána například statickou metodou `newUri()`, která díky tomu získá dostupné MIME typy a použije je pro naplnění obsahu objektu typu `ClipDescription`. S použitím Content Provideru již můžeme mluvit o P2P clipboardu, protože centralizovaná část clipboardu předá pouze informace o způsobu získání dat, ale data jsou předána pomocí nástroje meziprocesové komunikace Content Provider P2P. Architektura clipboardu OS Android je zachycena na následujícím obrázku:



Obrázek 3.4 Architektura Android clipboardu. [16]

3.7 Bezpečnostní analýza X.Org clipboardu

Bezpečnost ve spojitosti s X.Org je obecně vzato poměrně problematickou věcí. Vzhledem k tomu, že se nyní veškerá pozornost soustředí na nástupce tohoto systému Wayland, není bezpečnost X.Org prioritní problematika z pohledu vývojařů. Nenajdeme zde tedy moc bezpečnostních opatření ve prospěch clipboardu. Následující poznatky byly ověřeny také v praxi.

Každá aplikace, která chce pracovat s clipboardem, musí mít vytvořené okno. To ovšem není bezpečnostní opatření, které by mělo zajistit přístup ke clipboardu pouze aplikacím, jež neběží na pozadí. Okno totiž nemusí být aktivně používáno, ba dokonce nemusí být vykresleno. Musí existovat především z toho důvodu, aby bylo možné do vlastnosti okna zapsat předávaná data. Proto se dá s clipboardem pracovat velmi pohodlně z příkazového řádku pomocí nástroje `xclip`. Vzhledem k tomu, že clipboard manager si může neustále nechat posílat nová data clipboardu, může to tak dělat v podstatě libovolná aplikace a z hlediska uživatele to není možné zjistit. [24]

Logování, které by nějakým způsobem evidovalo předávání dat prostřednictvím clipboardu, bychom zde bohužel nenašli.

Vzhledem k tomu, že veškeré grafické aplikace zpravidla běží pod stejným uživatelem, má tento uživatel garantovaný přístup k X serveru. Ve chvíli, kdy má aplikace přístup k X serveru, může přečíst vlastnosti libovolného okna. Nezáleží tedy ani na tom, zda jsou tvůrci oken rozdílní. Aplikace, která má přístup k X serveru, může přečíst vlastnosti oken, které vytvořil například uživatel root. Pokud máme k dispozici ID okna, můžeme vypsat veškeré jeho vlastnosti. Xlib nám nabízí funkci `XListProperties()`, do které stačí jako parametr zadat ID okna a ona nám vrátí pole všech atomů identifikující veškeré vlastnosti okna. Následně stačí zavolat funkci `XGetWindowProperty()` s ID okna, jedním z atomů a dalšími parametry, které pro nás nejsou důležité. Funkce nám vrátí v jednom z parametrů data vlastnosti okna jako pointer na pole `unsigned char`. Pro zjednodušení existuje nástroj `xprop`, kterému stačí dát ID okna a on vypíše veškeré jeho vlastnosti. [19]

Pokud by byly implementovány nějaké bezpečnostní mechanismy clipboardu, jako například nějaká autentizace/kontrola, která by předcházela zapsání dat do vlastnosti okna, je problém v tomto nástroji meziprocesové komunikace. Než totiž dorazí k cílovému klientovi událost oznamující připravenost dat a než dojde k vyzvednutí dat a následnému zničení vlastnosti okna, je zde jisté časové okno na to, aby si tuto vlastnost vypsala aplikace, které tato data nenáleží.

Pokud bychom tedy shrnuli potenciální hrozby, budou následující:

1. monitoring clipboardu za účelem získání tajných dat (například hesel),
2. monitoring clipboardu a při pokusu o kopírování příkazu do terminálu uživatelem nahrazení příkazu v clipboardu za škodlivý.
   - V tomto případě se předpokládá scénář, ve kterém bude uživatel chtít vložit příkaz, který bude spouštět s rootovskými právy. Díky tomu by se tedy docílilo stavu, ve kterém by program běžící s právy uživatele, pomocí této techniky, dosáhl spuštění svého kódu s právy roota. Pro zvýšení pravděpodobnosti spuštění tohoto kódu uživatelem je možné využít techniku s vyčerpáním bufferu terminálu tak, jak jsme ji popsali v sekci 2.1.1.

Nyní si vyhodnotíme tyto hrozby pomocí metodiky CVSS:

Tabulka 3.1 Vyhodnocení hrozeb X.Org clipboardu.

Spuštění škodlivého příkazu		Únik tajných informací	
Základní skóre	Závažnost	Základní skóre	Závažnost
8,2	high	5	medium

Podrobnější hodnocení v příloze B.3 a B.4.

3.8 Bezpečnostní analýza Wayland clipboardu

To, jakým způsobem jsme si popsali fungování Wayland clipboardu, není zdaleka vše potřebné k tomu, abychom mohli pracovat s clipboardem. Jsou zde implementováná různá bezpečnostní opatření, která se snaží eliminovat práci s clipboardem, o které by uživatel nevěděl. Značnou část této problematiky jsme si popsali v kapitole 2. Veškerá následující analýza vychází nejen z literatury, ale také z výpisu komunikace mezi kompozitorem a klientem. Tuto komunikaci je možné monitorovat pomocí nastavení proměnné prostředí klienta na `WAYLAND_DEBUG=1`, tedy následující poznatky byly ověřené také v praxi. [23]

3.8.1 Vztah oknen a clipboardu

Základní opatření, které musí splňovat jak aplikace pro vkládání, tak pro kopírování, je existence okna. Okno musí být navíc nastaveno v roli toplevel, musí být vykresleno a musí být na něj zaměřeno, to znamená, že musí být aktivně používáno. Jedině v takovém případě budou klientovi chodit události spojené s clipboardem. Pokud je okno například minimalizované, události nechodí a žádosti spojené s clipboardem jsou ignorovány. O toto fungování se stará `xdg_shell`, který definuje role oken. [22][22][25]

Nejprve je nutné vytvořit okno. To se provede tak, že se vytvoří objekt `xdg_surface` pro existující povrch reprezentován objektem `wl_surface`. Role toplevel je oknu přiřazena tak, že se promapuje objekt `xdg_surface` s objektem `xdg_toplevel`, který je nutné vytvořit. Pokud je okno z minimalizace opět zobrazeno, smyčka událostí opět začne fungovat a první událost, která odstartuje opět ostatní události, je kontrola okna pomocí události `ping()` spojené s globálním rozhraním `xdg_shellu`. Na tuto událost se musí odpovědět žádostí `pong()`, který musí obsahovat sériové číslo události `ping()`. [22][5][25]

Toto opatření má zřejmě zamezit používání clipboardu v aplikacích běžících na pozadí. Problém je ovšem v tom, že při spuštění jakékoliv aplikace s oknem je fungování takové, že okno je zobrazeno přes všechny ostatní aktuální okna a je zaměřené. Pokud tedy uděláme dostatečně malé okno 1x1 pixelů, uživatel ho nezaznamená a získáme tím jednorázový přístup ke clipboardu, čili můžeme vložit data z clipboardu. Pokud vykreslíme malé okno, které uživatel nevidí, uvidí ale na hlavním panelu existenci nějaké minimalizované aplikace. Toto můžeme nicméně také obejít. Vzhledem k tomu, že vykreslení okna trvá nějakou dobu a propsání na hlavní panel také, je možné okno zkonstruovat, přijmout data z clipboardu a posléze zničit. Okno se stihne vykreslit jen na krátkou chvíli a nestihnou se propsat informace na hlavní panel, tím pádem uživatel nezaznamená, že nějaké okno vůbec existovalo. Pokud bychom volali ve smyčce tedy zkonstruování okna, přijetí dat z clipboardu a následnou destrukci okna, mohlo by opět dojít k nekontrolovanému unikání dat z clipboardu.

Naštěstí nově vytvořené okno znamená, že pokud používáme nějaké jiné okno, přestane být zaměřené, protože zaměřené bude nově vytvořené okno. Tím pádem pokud poběží tento program a my budeme chtít něco zkopírovat ve stávajícím okně, najednou to nepůjde a celý systém se bude chovat nestabilně a na pohled bude viditelné, že něco není v pořádku. Pokud bychom tento útok provedli tím způsobem, že bychom skenování clipboardu prováděli například každou hodiny, nemusel by to uživatel zaznamenat. Nicméně je tím alespoň snížena pravděpodobnost uniknutí nějaké tajné informace, pokud heslo z clipboardu včas vymažeme. Bohužel explicitní notifikaci ohledně podezření klienta z důvodu častého přístupu ke clipboardu neuvidíme.

3.8.2 Kopírování a triviální autentizace

Kopírování má kromě zabezpečovacího mechanismu, který je popsán v sekci 3.8.1, implementováno triviální autentizaci. Ta spočívá v tom, že žádost `set_selection()`, která je spojena

s objektem `wl_data_device`, musí obsahovat sériové číslo události, jež akci kopírování vyvolala. Kompozitor pak rozhoduje, zda žádost klienta přijme nebo ne. [22][5]

Problém této metody je aktuálně v tom, že jako událost akceptuje i `wl_keyboard::enter()`, která pouze říká, že focus klávesnice je na okně, které chce kopírovat. Což je de facto znovu potvrzení toho, o co již se stará `xdg_shell`, jak bylo popsáno v sekci 3.8.1. Abychom byli korektní, není to problém této metody, spíše implementace v kompozitorech. Tato skutečnost byla v rámci této práce ověřena jak na kompozitoru Weston, tak na kompozitoru KWin. [22][5]

3.8.3 Clipboard manager a Wlr_data_control

Wlr_data_control protokol není standardním protokolem od tvůrců Waylandu. Jedná se o externí protokol, který má sloužit pro komunikaci clipboardu a clipboard manageru. Implementuje ho například kompozitor KWin. Vzhledem k bezpečnostním opatřením, které používá Wl_data protokol, by nebylo možné implementovat clipboard manager, který by používal tento protokol. Clipboard manager musí mít možnost získávat obsah od klientů bez nutnosti mít vlastnictví nějakého zaměřeného okna. [22]

Proto byl vytvořen tento protokol, který umožňuje monitorování clipboardu. Tento protokol de facto odbourává veškerá bezpečnostní opatření. Používá se podobně jako Wl_data protokol s tím, že není nutné vytvářet vůbec okno a události chodí klientovi, který tento protokol používá, neustále. Tento protokol je navíc kompatibilní s klienty, kteří používají Wl_data protokol. Například klient, který data kopíruje, může používat Wl_data protokol a clipboard manager, který chce data vložit, použije Wlr_data_control protokol. Počet clipboard managerů není nějak regulován, tudíž libovolná aplikace na pozadí může použít tento protokol k tomu, aby si nechala zasílat data z clipboardu bez vědomí uživatele. Součástí přílohy je jednoduchá aplikace, která vypisuje data z clipboardu pokaždé, co se jeho obsah změní. Jedná se o proof-of-concept tohoto problému. [22]

Tento protokol také umí pracovat s primary_selection clipboardem. Tím pádem, pokud existuje v systému aplikace, která používá tento protokol, má k dispozici veškerá data, která uživatel zkopíruje a veškerý text, který uživatel pouze označil. [22]

Dalším problémem je clipboard manager jako takový, jak jsme si popsali v sekci 2.2. U Klipperu (defaultního clipboard manageru prostředí KDE Plasma) je možné získávat jeho data pomocí API a navíc je možné číst soubor, do kterého je obsah ukládán v otevřeném textu. Většina clipboard managerů v Linuxových OS evidují obsah do souborů, které jsou uložené na pevném disku zpravidla někde v `~/.cache`. V terminálu lze pro získání dat z Klipperu použít následující příkazy:

Výpis kódu 3.1 Příkazy pro práci s managerem Klipper. [7]

```
$ qdbus org.kde.klipper /klipper  ...
   org.kde.klipper.klipper.getClipboardContents    -- QString()
   org.kde.klipper.klipper.getClipboardHistoryItem -- QString(int i)
   org.kde.klipper.klipper.getClipboardHistoryMenu -- QStringList()
   org.kde.klipper.klipper.saveClipboardHistory    -- void()
   org.kde.klipper.klipper.setClipboardContents    -- void(QString s)
```

Důležité je zmínit, že Klipper, a obecně clipboard manager, obsahuje jenom taková data, jaká mu byla poskytnuta. Na jeho výzvy o zaslání dat mohou aplikace reagovat tím způsobem, že data neposkytnou. Nicméně to není v praxi zatím využíváno.

Nyní si vyhodnotíme opět závažnost následujících dvou hrozeb:

1. únik tajných dat,
2. spuštění škodlivého kódu jako root změnou obsahu clipboardu.

Závažnost těchto hrozeb budeme určovat v závislosti na používaném protokolu. Přidáme zde také hodnocení Klipperu, které se týká jak Waylandu tak X.Org.

Tabulka 3.2 Vyhodnocení hrozeb Waylandového clipboardu.

Místo hrozby	Spuštění škodlivého příkazu		Únik tajných informací	
	Základní skóre	Závažnost	Základní skóre	Závažnost
Wl_data protokol	8,2	high	5,9	medium
Wlr_data_control protokol	8,2	high	5,9	medium
Klipper	8,2	high	5.9	medium

Podrobnější hodnocení v příloze B.6 a B.7.

3.9 Bezpečností analýza androidového clipboardu

V OS Android je bezpečnost clipboardu poněkud lepší, než tomu je u desktopových OS. Aby se docílilo zamezení přístupu ke clipboardu aplikacím bez vědomí uživatele, je tedy metoda `getPrimaryClip()` opatřena několika bezpečnostními mechanismy. Jedním z nich je neposkytnutí obsahu clipboardu, pokud aplikace nemá buď fokus na zadávání vstupu, nebo není defaultním editorem metody zadávání (jako jsou například vitruální klávesnice). To by mělo zabránit aplikacím, které nejsou aktivně používány, manipulaci s clipboardem. [16]

Dále je tato metoda od verze Androidu 12 a vyšší vybavena notifikací v podobě banneru, který uživatele informuje, že aplikace vložila obsah z clipboardu. Díky tomu nemůže žádná aplikace zavolat tuto metodu bez toho, aby o tom uživatel věděl. [16]

Problém ovšem může nastat tehdy, pokud uživatel zkopíruje heslo nebo tajná data do clipboardu, následně vloží tato data do cílové aplikace a obsah neodstraní z clipboardu. Potom může začít pracovat s jinou aplikací a začne psát například do jejího textového pole. Ta může v tuto chvíli data z clipboardu získat. Uživatel se to sice doví bannerem, ale k úniku dat již došlo.

Clipboard manager je zpravidla implementován jako jeden z nástrojů virtuální klávesnice. Zde dochází k monitorování clipboardu a evidování historie clipboardu. Pokud používáme klávesnici z nedůvěryhodného zdroje, může se jednat o klávesnici, která je spywarem cíleným na clipboard. [16]

Pokud bychom měli ohodnotit možnost úniku tajných dat z clipboardu metodikou CVSS, bude výsledek následující:

Tabulka 3.3 Vyhodnocení hrozeb clipboardu OS Android.

Únik tajných informací	
Základní skóre	Závažnost
5	medium

Podrobnější hodnocení v příloze B.5.

Kapitola 4

Možnosti vylepšení bezpečnosti X.Org a Android clipboardu

V této kapitole jsou popsány možnosti, jak vylepšit bezpečnost clipboardových implementací X.Org a Android, a také zmiňuje vhodná místa pro umístění kontrolních mechanismů zajišťujících větší bezpečnost implementací.

4.1 Možnosti X.Org clipboardu

V tuto chvíli by asi nemělo smysl podrobně popisovat, jakým způsobem reimplementovat X server, aby clipboard fungoval bezpečně, neboť se jedná o zastaralý software, který bude brzy nahrazen Waylandem. Wayland kompozitor můžeme de facto považovat za reimplementaci X serveru, proto řešení, v rámci kterého by bylo nutné reimplementovat X server, může již existovat ve Waylandu. Za této situace je nejjednodušším řešením na Wayland přejít. Už v tuto dobu je možné Wayland používat, nicméně X.Org je stále výchozí volba většiny systémů, proto si ukážeme možnosti, které nabízí X.Org API a jsou využitelné pro bezpečnější práci s clipboardem.

Existuje knihovna libXRes, která je rozšířením klihovny Xlib. Většina Linuxových OS, které používající X.Org, tuto knihovnu zahrnuje. Ta obsahuje funkce, které umožňují pomocí na základě ID okna získat PID procesu, který okno vytvořil. Toto by se dalo použít pro autentizaci protistrany před tím, než klient poskytující data je zapíše do vlastnosti okna protistrany. Je ovšem otázka, jestli by tato autentizace byla vůbec relevantní, neboť libovolný klient může číst vlastnosti libovolných oken. To, že bychom zjistili PID procesu, který toto okno vlastní, by neznamenalo, že cílová data skončí pouze v adresním prostoru tohoto procesu.

Proto by bylo lepším řešením, kdyby tento mechanismus clipboardu předal například FD k nějaké rouře, která by byla použita posléze k přenesení dat. Tento model již by se podobal tomu, co je používán ve Waylandu. Metody zajištění bezpečnost tohoto přenosu jsou popsány podrobněji v sekci 5.2. Toto řešení by ovšem fungovalo pouze mezi lokálními klienty.

Řešení, které by zamezilo modifikaci obsahu clipboardu, se bohužel týká reimplementace X serveru, neboť řízení přístupu ke clipboardu je v jeho režii.

4.2 Možnosti Android clipboardu

Reimplementace clipboardu na straně OS Android by byla určitě možnost, která by dokázala problém efektivně vyřešit. Nejjednodušší by bylo změnit informační banner, který oznamuje vložení dat z clipboardu, na banner, ve kterém by uživatel musel explicitně odsouhlasit poskytnutí

clipboardových dat aplikaci, která o ně žádá. Díky tomu by byl i centralizovaný model clipboardu dostatečně bezpečný. Další možnosti, které jsou vhodnými kandidáty pro implementaci na straně clipboardu/serveru, si uvedeme v sekci 5.1 na konkrétním příkladu Wayland kompozitoru.

Bez nutnosti reimplementace clipboardu je rozumnou volbou neposkytovat citlivý obsah prostřednictvím centralizovaného modelu clipboardu, ale použít Content Provider, tedy P2P model clipboardu. Vzhledem k tomu, že pro tvorbu Content Provideru je nutná implementace metody `query()`, která má poskytnout data při zavolání protistrany, je toto místo vhodným kandidátem na umístění autentizace a bezpečnostních mechanismů, které zajistí bezpečné předání obsahu. Docílili bychom tím i eliminaci clipboard manageru, který by neevidoval data, která nabízíme. Více o bezpečnostních mechanismech, které by byly uplatnitelné i pro Android clipboard si řekneme v sekci 5. [16]

Kapitola 5

Bezpečnější Wayland clipboard

Tato kapitola obsahuje návrhy řešení zranitelností Waylandového clipboardu a rozbory jejich silných a slabých stránek. Cílem těchto návrhů je umožnit bezpečnější používání Waylandového clipboardu. Dále jsou zde diskutovány možnosti nasazení těchto návrhů. Zvláště podrobně je zde popsáno řešení, které bylo zvoleno pro implementaci proof-of-conceptu v jazyce C/C++. Také je zde ukázáno použití implementovaného proof-of-conceptu a jeho zakomponování do správce hesel.

5.1 Řešení na straně kompozitoru

Cílem řešení na straně kompozitoru je zabránit poslání událost `send()` klientovi, který poskytuje data. Jak již víme, událost `send()` je vytvořena a odeslána v reakci na žádosti `receive()`, kterou zasílá kompozitoru klient poptávající data z clipboardu. Vzhledem k tomu, že tato dvojice žádost–událost existuje napříč všemi protokoly, jež manipulují s clipboardem, a používá se stejným způsobem, je zamezení poslání události `send()` správný cíl. Jak víme, bezpečnostní opatření na straně kompozitoru ohledně odmítnutí poskytnutí přístupu ke clipboardu nejsou dostatečná. [5]

Komunikace mezi klientem a kompozitorem je v protokolu Wayland implementována pomocí socketů typu UNIX domain socket. Proto bychom zřejmě mohli využít nástrojů, které nabízí UNIX domain socket, k získání identity klienta, který chce obsah clipboardu. Díky těmto informacím je možné provést autentizační kroky (například prohledání blacklisů/whitelistů) a podle výsledku autentizace poslat resp. neposlat událost `send()`.[5]

Ve Wayland server API se nachází následující funkce, se kterou je možné získat informace o klientovi:

```
void wl_client_get_credentials(struct wl_client *client, pid_t *pid,
                               uid_t *uid, gid_t *gid)
```

Můžeme si všimnout, že funkce vrací pid, uid a gid. Tyto informace nám určitě stačí k tomu, abychom získali název programu včetně absolutní cesty ke spustitelnému souboru (například přečtením symbolického odkazu `/proc/pid/exe`). Nicméně situace je poněkud komplikovanější. Tato funkce používá volání jádra `getsockopt()` s parametrem `SO_PEERCRED`. Problematiku s touto metodou si popíšeme v následující sekci. [5][26][27]

5.1.1 UNIX domain socket – credential

UNIX domain socket dovedou, kromě posílání běžných dat, předávat doplňkové zprávy. Na posílání a přijímání těchto zpráv slouží volání jádra `sendmsg()` a `recvmsg()`. Jedna z podmnožin

těchto zpráv nese název `SCM_CREDENTIALS`. Jedná se o identifikátory procesu a jsou definovány následující strukturou:

```
struct ucred{pid_t pid; uid_t uid; gid_t gid;};
```

Tento typ dodatečné zprávy je kontrolován přímo jádrem operačního systému Linux. To znamená, že proces nemůže poslat falešné identifikátory, pokud se nejedná o privilegovaný proces. Jak jsme již zmínili na začátku sekce 5.1, funkce nabízena Waylandovým server API udělá de facto tuto operaci:

Výpis kódu 5.1 Operace getsockopt a získání credential.

```
void wl_client_get_credentials(int socket_fd,pid_t *pid,
                               uid_t *uid, gid_t *gid)
{
   struct ucred cred_peer;
   socklen_t cred_peer_len = sizeof(cred_peer);
   getsockopt(socket_fd, SOL_SOCKET, SO_PEERCRED,
              &cred_peer, &cred_peer_len);
   *pid=cred_peer.pid;
   *uid=cred_peer.uid;
   *gid=cred_peer.gid;
}
```

Toto řešení je velmi jednoduché, ale má poměrně zásadní nedostatek pro některé případy. Metoda SO_PEERCRED, kterou toto řešení zahrnuje, funguje tak, že jsou tyto údaje vygenerovány při zavolání `connect()`, kterým se klient připojuje k socketu, a poté se již nezmění. [26]

Nyní si ukážeme zkrácený výpis z kódu klienta v příloze A.2, který jsem vytvořil pro lepší pochopení problematiky. Vidíme, že se zde používá volání jádra `fork()`, které rozdvojí původní proces na dva procesy v rolích rodič a syn. Rodičovský proces má stejné PID, synovskému procesu je přiděleno nové PID. Synovský proces zdědí množinu otevřených filedescriptorů od svého rodiče. Je-li mezi otevřenými FD i takový, který odkazuje na zapisovací konec socketu, má k němu synovský proces přístup. Spustí-li voláním `exec()` nepřijatelný program, má tento program možnost zapisovat do socketu (pokud nemá filedescriptor socketu nastaven flag close-on-exec).

Server nepozná, zda do FD zapisuje rodičovský nebo synovský proces, protože PID bylo do dodatečné zprávy obsahující credentials zapsáno při volání `connect()`. Procesem, který se připojoval k socketu, byl rodičovský proces, proto bude server při identifikaci připojeného klienta vždy dostávat PID rodičovského procesu, a tudíž nezáleží na tom, kdo socket používá.

Výpis kódu 5.2 Operace fork() po vytvoření socketu

```
int ret = write(data_socket,"123456789",10);
pid_t fork_ret=fork();
if (fork_ret<0) return -1;
else if (fork_ret==0)
{
   /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */
    ret = write(data_socket,"123456789" ,10 );
    if (ret == -1)
        exit(EXIT_FAILURE);
    return 0;
}
else {
   int dummy;
 waitpid(fork_ret, &dummy, 0);
 }
```

Zvýšenou pozornost věnujme části kódu s vykřičníky. Je důležité vědět, že při volání `write()` za vykřičníky již zapisuje do socketu naprosto jiný proces (někdo jiný). Zavoláním funkce, kterou jsme si definovali ve výpisu kódu 5.1, dostaneme vždy PID rodiče, nikoli zapisovatele do socketu.

5.1.2 SO_PASSCRED a jeho benefity oproti SO_PEERCRED

SO_PEERCRED jsme si již vysvětlili v sekci 5.1.1, nyní si vysvětlíme druhou variantu získání credentials a srovnáme si tyto dvě varianty.

Tato varianta se aktivuje na socketu pomocí parametru SO_PASSCRED. K nastavení tohoto parametru je možné použít volání jádra `setsockopt()`. Poté je možnost získat credentials při každém zapisování klienta do socketu. Nejprve přečteme dodatečná data obsahující credential a po ověření jeho totožnosti můžeme pomocí volání `read()` přijmout již data, která očekáváme. [26]

SO_PASSCRED má velkou výhodu v tom, že při každém přijímání dat můžeme autentizovat klienta a víme, kdo nám data posílá. Pro lepší vysvětlení problematiky si ještě ukážeme část z kódu jednoduché serverové aplikace, který je k dispozici v příloze A.1.

Výpis kódu 5.3 Autentizace klientů SO_PEERCRED vs SO_PASSCRED

```
int data_socket=... /*zde bude hodnota
filedescriptoru po skonceni accept()*/
struct msghdr msgh;
msgh.msg_name = NULL;
msgh.msg_namelen = 0;
struct iovec iov;
int data;
msgh.msg_iov = &iov;
msgh.msg_iovlen = 1;
iov.iov_base = &data;
iov.iov_len = sizeof(data);
union {
  char buf[CMSG_SPACE(sizeof(struct ucred))];
  struct cmsghdr align;
} controlMsg;
msgh.msg_control = controlMsg.buf;
msgh.msg_controllen = sizeof(controlMsg.buf);
struct cmsghdr *cmsgp;
int optval = 1;
if (setsockopt(data_socket, SOL_SOCKET, SO_PASSCRED, &optval,
               sizeof(optval)) == -1) {
  clean(data_socket, connection_socket);
  exit(EXIT_FAILURE);
}
recvmsg(data_socket, &msgh, 0);
cmsgp = CMSG_FIRSTHDR(&msgh);
memset(&rcred, 0, sizeof(rcred));
if (cmsgp->cmsg_level != SOL_SOCKET
    || cmsgp->cmsg_type != SCM_CREDENTIALS) {
  clean(data_socket, connection_socket);
  exit(EXIT_FAILURE);
}
memcpy(&rcred, CMSG_DATA(cmsgp), sizeof(struct ucred));
printf("SO_PASSCRED pid=%ld, uid=%ld, gid=%ld\n",
       (long)rcred.pid, (long)rcred.uid, (long)rcred.gid);
```

```
if (read(data_socket, buffer, sizeof(buffer)) == -1) {
  clean(data_socket, connection_socket);
  exit(EXIT_FAILURE);
}
```

Tento kód je poněkud delší než kód 5.1, protože zpráva se musí rozparsovat správným způsobem, abychom dostali finální strukturu `ucred`. V příloze A.1 je obsažen celý kód serveru i s výstupem, který server vypíše, když se společně s ním spustí kód klienta (ten se nachází v příloze A.2). Je zde dobře viditelné, jakým způsobem se liší PID před voláním `fork()` a po něm. [26]

Problém metody SO_PASSCRED je v tom, že nemůžeme získat credentials hned po volání `accept()`, ale až po prvním zapsání pomocí volání `write()`. Proto je rozumnou variantou používat kombinaci obou metod, ale každou v tom místě, kde přináší největší benefity. Na otázku, jestli má smysl používání metody SO_PASSCRED a nebylo by jednodušší používat metodu SO_PEERCRED se současným prověřováním všech synovských procesů, je odpověď jednoznačně ne. Je totiž možné předat filedescriptor pomocí dalšího socketu nebo zduplikovat FD pomocí volání jádra do aplikaci, která není synem, a tím by získala přístup k tomuto socketu opět pod údaji aplikace, která uzavřela spojení. [26]

5.1.3 Kontrolní mechanismy a finální podoba řešení

Pokud bychom chtěli, aby měl klient možnost se rozhodnout, zda poskytne data protějšku, bylo by nutné reimplementovat protokol. Server by mu musel předat v nějaké události credentials protistrany, aby měl klient dostupná data, na základě kterých by mohl rozhodnout. Jako další by následovala žádost obsahující rozhodnutí klienta.

Pokud bychom nechtěli pozměňovat protokol na předávání dat clipboardu nebo definovat vlastní, znamená to jisté omezení z hlediska kontrolních mechanismů, které máme k dispozici, nicméně nabízejí se nám tyto:

- odmítnutí přístupu klienta ke clipboardu v případě, že není uveden ve whitelistu nebo je uveden v blacklistu,
- neposkytnutí obsahu clipboardu klientovi na základě ACL,
- logování komunikace s clipboardem,
- monitorování provozu a následní blokování přístupu ke clipboardu:
  1. při příliš frekventovaném dotazování na obsah clipboardu,
  2. při příliš frekventované snaze o kopírování dat.

K autentizaci klientů jsou uplatnitelné námi popsané principy v sekci 5.1.2 s tím, že Waylandové server API by se zřejmě rozšířilo, z důvodu snazšího používání, o možnost získání credentials metodou SO_PASSCRED. Na základě získané identity (PID a z něho odvoditelného unikátního jména aplikace) by byla provedena autorizace, ve které by se rozhodlo, zda má klient právo manipulovat s clipboardem. Autentizaci je nutné dělat při každé žádosti klienta, abychom poznali případnou změnu identity.

Autorizace klienta by probíhala pomocí blacklistů a whitelistů. Na základě nich by došlo k eliminováni klientů, kteří nemají mít přístup ke clipboardu vůbec.

Každá manipulace s clipboardem by byla logována. Rozhodnutí, zda budou data poskytnuta, by se odvíjelo od ACL. V ACL by bylo definováno, jaké aplikace si smějí vyměňovat společně clipboardová data.

Monitorování komunikace by evidovalo četnosti pokusů o kopírování/vkládání dat u jednotlivých aplikací. Pokud by nějaká aplikace vykazovala chování spywaru, kompozitor by to reflektoval v logování a zablokoval by přístup ke clipboardu takové aplikaci.

Nyní si popíšeme vhodná místa, kam tyto kontrolní mechanismy umístit. Kontrolu ACL, zalogování pokusu o vložení dat a monitorování četnosti pokusů o vložení je vhodné umístit do následujícího místa v kódu kompozitoru Weston, v případě obecného kompozitoru se jedná o callback reagující na žádost `receive()`:

Výpis kódu 5.4 Callback reagující na receive(): weston/libweston/data-transfer.c

```
static void data_offer_receive(struct wl_client *client,
                               struct wl_resource *resource,
                               const char *mime_type, int32_t fd)
{
   struct weston_data_offer *offer =
          wl_resource_get_user_data(resource);
   if (offer->source && offer == offer->source->offer /*and somecheck()*/)
      offer->source->send(offer->source, mime_type, fd);
   else
      close(fd);
}
```

Důvod je ten, že až po poslání žádosti `receive()` jednoznačně víme, že protistrana poptává obsah od zdrojového klienta.

Monitorování komunikace by bylo vhodné umístit na více míst, stejně tak kontrolu na základě blacklistů/whitelistů. Vhodná místa by byla:

- žádost `create_data_source()`,
  - V callbacku reagujícím na tuto žádost je vhodné monitorovat četnost pokusů o kopírování jednotlivých klientů a dále neumožnit vytvoření objektu `wl_data_source` při častých pokusech o kopírování, pokud je klient uveden v blacklistu nebo pokud klient není uveden ve whitelistu.
- žádost `get_data_device()`,
  - V callbacku reagujícím na tuto žádost je vhodné monitorovat četnost pokusů o získání `wl_data_device`, díky kterému je možné data vkládat nebo kopírovat. V reakci na příliš vysokou četnost by kompozitor neposkytnul objekt `wl_data_device`, bez kterého není možné vkládání ani kopírování. Pokud by se klient nacházel v blacklistu nebo by se nenacházel ve whitelistu, chování by bylo stejné, tedy neposkytnutí objektu.
- žádost `receive()`.
  - Callback reagující na tuto událost by umožňoval monitorovat četnost pokusů o vložení a v reakci na příliš časté pokusy by kompozitor přestat posílat události spojené s clipboardem podezřelému klientovy.
  - Zde by bylo vhodné ještě navíc cílit s monitorováním klientů požadujících vložení dat z clipboardu ve vztahu ke klientovi poskytujícímu data.

5.1.4 Problémy spojené s nasazením řešení

Pokud bychom řešili bezpečnější chování clipboardu na straně kompozitoru, jako první problém je především to, že se musí reimplementovat části kompozitoru, případně celý protokol. Kompozitor je poměrně velký projekt a je důležité, aby fungoval rychle. Proto by bylo nutné se zabývat otázkou, zda čekání na autentizaci nezpůsobí problémy v hlavní smyčce událostí, neboť řešení, které by bylo sice bezpečnější, ale na úkor plynulého renderingu oken, by bylo nepřijatelné. Kompozitorů je mnoho, je otázka, zda by se tento typ řešení dokázal prosadit alespoň do

těch nejpoužívanějších. Formát blacklistů, whitelistů a access control listů by bylo třeba standardizovat, aby nelišil napříč kompozitory. Protože různí tvůrci kompozitorů se dívají na stejnou problematiku různými pohledy, zřejmě i formát access control listů by nebyl výjimkou.

Reimplementace protokolu by byla velký problém, protože všechny grafické aplikace pracují s clipboardem aktuálně dostupným způsobem a tento přístup by musely změnit. Tuto zásadní změnu by se opět prosazovalo velmi těžko. Byla by určitě možnost udělat nový protokol, který by byl kompatibilní se stávajícím protokolem (to by mohla zajistit již konrkétní implementace protokolu na straně kompozitoru). Nicméně pokud by kvůli zpětné kompatibilitě existovala možnost, jak používat clipboard nebezpečnějším nýbrž jednodušším způsobem, vývojaři často upřednostní tuto variantu.

5.1.5 Silné a slabé stránky řešení

Výhodou tohoto řešení je, že je možné udělat komplexnější systém ověřování a bezpečnostních kontrol na více úrovních protokolu. Značná část navrhovaných bezpečnostních mechanismů by navíc neovlivnila to, jakým způsobem klienti používali clipboard doposud. Monitoring přistupování ke clipboardu je velký benefit tohoto řešení, který asi jinde než na straně kompozitoru není možné efektivně nasadit.

Značná část problémů tohoto řešení již byla popsána v sekci 5.1.4. Nevýhodou tohoto řešení je příliš velká složitost vzhledem k tomu, že se jedná o triviální nástroj. Pokud bychom chtěli možnost rozhodnutí nechat na klientovi, který poskytuje data ke kopírování, je nutnost reimplementace protokolu velmi nežádoucí faktor a vznáší to otázku, zda by nebylo jednodušší řešit tento problém na straně klienta bez změny protokolu.

Autentizační mechanismy UNIX domain socket jsou efektivní v tom, že máme možnost získat údaje o protějšku bez nutnosti kontroly všech filedescriptorů aktuálně běžících procesů. Nicméně zůstává otázka, jestli by nebylo vhodné zkontrolovat datovou strukturu, na kterou odkazuje filedescriptor předaný v žádosti `receive()`. Klient sice může být důvěryhodný, ale může se stát, že vytvoří jinou datovou strukturu než rouru, která je v protokolu požadována, případně že s ní nebude pracovat správně. Tato problematika je rozebrána podrobněji v sekci 5.2.1.

5.2 Řešení na straně klienta

Toto řešení je navrženo tak, že veškerá zodpovědnost za bezpečné doručení z aplikace, která data poskytuje, do aplikace, která data přijímá, je na aplikaci poskytující data. Toto řešení jsem se také rozhodl implementovat, a tudíž je součástí přílohy zdrojový kód včetně dokumentace. Smyslem řešení je, aby nebylo nutné měnit aktuální implementace kompozitorů a aby nebylo nutné měnit protokol na práci s clipboardem. Toto řešení má zprostředkovat možnost bezpečného předání dat pro klienty, které tuto možnost potřebují (jako jsou například správci hesel).

V sekci 3.5.1 jsme si detailně popsali, jakým způsobem funguje protokol wl_data. Vzhledem k tomu, že je aktuálně nejrozšířenšjí, je toto řešení postaveno na něm. Myšlenka řešení je poměrně jednoduchá. Víme, že po příchodu události `send()`, která je spojená s objektem `wl_data_source`, následuje zapsání dat do FD, který je obsažen v příchozí události a nám je předám v parametru callbacku, který reaguje na tuto událost. My, místo okamžitého zapsání dat v požadovaném formátu, můžeme v tomto callbacku provést mechanismy kontrol. Například zkontrolovat datovou strukturu, na kterou FD odkazuje, a zjistit, jaké všechny aplikace mají přístup k této struktuře prostřednictvím svých FD.

Víme, že datová struktura by měla být roura, kterou vytvořil klient požadující data. My tedy potřebujeme zjistit minimálně PID procesu, který je na druhém konci roury, abychom jej mohli identifikovat.

5.2.1 Problematika rour a file descriptorů

Identifikace datové struktury, na kterou odkazuje FD, není nic složitého. Vzhledem k tomu, že došlo k předání FD prostřednictvím UNIX domain socket, patří FD našemu procesu. Potřebujeme, aby náš systém měl přístup do proc filesystemu. Ten by měl být funkční ve všech konvenčních Linuxových systémech a přimontovaný v `/proc`. Tento virtuální filesystém obsahuje informace o všech běžících procesech. Chceme-li získat informace o všech file descriptorech našeho procesu, potřebujeme jeho PID. Jeho zjištění nám zajistí volání jádra `getpid()` obsažené v hlavičce `unistd.h`.

Po jeho získání se můžeme podívat do adresáře `/proc/[získane_PID]/fd/`, který obsahuje všechny otevřené FD tohoto procesu. FD jsou zde reprezentovány jako soubory, jejichž jménem je číslo odpovídající file descriptoru. Tyto soubory nicméně nejsou obyčejné, jedná se o symbolické odkazy. Symbolické odkazy je možné přečíst a tím zjistit cíl, na který odkazují. Dá se to udělat například pomocí tohoto volání jádra:

```
#include <unistd.h>
ssize_t readlink(const char *restrict pathname, char *restrict buf,
                 size_t bufsiz);
```

případně je možné použít C++ nadstavbu nad tímto voláním z STL `std::filesystem`. Číslo FD známe z události, stačí tedy přečíst `/proc/fd/[získane_PID]/fd/[FD_z_události]`. Výstup by mohl vypadat například takto: `pipe:[77720]`. Číslo v závorkách je unikátní identifikátor roury, neboť se jedná o i-node v pipefs (virtuální filesystém obsahující roury). [27]

V případě, že je výstupem jiná strukturu, můžeme jednoduše FD zavřít pomocí volání `close()` a žádná data neposílat, neboť protistrana vykazuje podezřelé chování nedodržením protokolu.

Dalším důležitým krokem je zjištění PID všech procesů, které na tuto rouru odkazují. To se dá udělat jenom tím způsobem, že projdeme všechny procesy a podíváme se, na jaké datové struktury odkazují jejich file descriptory. Ve chvíli, kdy narazíme na shodu, tedy na FD, který odkazuje na rouru se stejným i-node, poznamenáme si PID tohoto procesu (iterujeme přes všechny symbolické odkazy, jejichž umístění odpovídají regulárnímu výrazu: `/proc/[0-9]+/fd/[0-9]+`).

Pokud nalezneme jiný počet shod než dvě, nepošleme data, neboť nemůžeme zajistit, že data dojdou do správné aplikace. U procesů běžících pod jiným uživatelem než naším nebude možné číst informace o FD. Nicméně to není problém. Protože pokud by existoval proces, který by dokázal běžet pod jiným uživatelem než naše aplikace poskytující data, a zároveň by dokázal zařídit to, aby nám počet shod vyšel přesně dvě, zřejmě se jedná o proces s vysokými oprávněními. Pokud by proces měl nízká oprávnění a běžel pod jiným uživatelem, nebyl by schopen vytvořit synovský proces pod stejným uživatelem, který data poskytuje. Toto je privilegovaná operace. V tomto případě je získání nějakých dat z clipboardu naprosto marginální k možnostem, jaké tento proces má. Tato obrana cílí především na aplikace, které běží pod stejným uživatelem.

Následně již máme PID protějšku a můžeme zjistit o procesu více informací, například absolutní cestu ke spustitelnému souboru. Poté je možné prohledat blacklisty a whitelisty, zda se zde proces nenachází, případně se dotázat uživatele, zda chce tomuto procesu poskytnout data. Tato činnost trvá poněkud delší dobu. Proto je nutné těsně před tím, než data zapíšeme, znovu zkontrolovat, zda jsou k rouře přidružené stále tytéž dva procesy. Důvod nám nejlépe demonstruje tento program:

Výpis kódu 5.5 Přesměrování roury z jiného procesu na sebe.

```
#include<stdio.h>
#include<string.h>
#include<sys/syscall.h>
#include<unistd.h>
#include<signal.h>
int main(void)
```

```
{
   pid_t pid=...
   int fd=...
   int pidfd = syscall(SYS_pidfd_open,pid,0);
   int newfd = syscall(SYS_pidfd_getfd,pidfd,fd,0);
   kill(pid, SIGKILL);
   char buffer[2048];
   while (read(newfd, buffer, 2047)>0)
   {
      printf("Data %s\n",buffer);
      memset(buffer, 0, sizeof(buffer));
   }
}
```

Tento program nejprve otevře FD odkazující na běžící proces, jehož PID zadáme, posléze voláním `SYS_pidfd_getfd()` duplikuje FD z procesu, jehož PID jsme zadali. Tento FD může odkazovat na čtecí konec roury, kterou budeme posílat například heslo ze správce hesel. Následně je původní proces, který byl vlastník čtecího konce roury, zabit signálem `SIGKILL` a jediný, kdo čeká na obsah z roury, je tento škodlivý program. [28]

Toto je možné udělat, pokud vše běží pod jedním uživatelem, což je náš případ. Jiný uživatel totiž nemá přístup k file descriptorům procesů, které nevlastní. Jediný, kdo má tato práva, je root. Běží-li škodlivý program s právy roota, není až tak velkou prioritou mít v tuto chvíli efektivní ochranu clipboardu, neboť s takto vysokými právy může systému uškodit mnohem více a sofistikovaněji.

Po druhé kontrole roury musíme okamžitě zapsat data do roury. Cílem je nechat časové okno a možný vektor útoku co nejmenší.

5.2.2 Blacklist a whitelist

Pro tvorbu blacklistů nebo whitelistů je třeba nějaký unikátní identifikátor popisující program, který byl spuštěn. Aby bylo jednoduché tyto ACL vytvořit, nabízí se jméno aplikace. Nicméně je důležité, aby toto jméno bylo unikátní. Nejpřímočařejším postupem by zřejmě bylo zkontrolovat jméno procesu v `/proc/[pid]/comm`, případně `/proc/[pid]/cmdline`. Tento postup by byl ovšem fatální chybou, neboť cmdline nemusí obsahovat reálné jméno procesu. Toho se dá docílit tak, že hodnotu argumentu, do kterého se běžně ukládá jméno programu, nastavíme na libovolné jméno. Posléze je zavolán `execve()`, kterému předáme argumenty zahrnující nepravdivé jméno programu, a tudíž se spustí program s nepravdivým cmdline. [27]

Comm i cmdline jsou problematické také v tom, že se jedná pouze o krátký název programu. Při spuštění binárního souboru `/bin/sleep` uvidíme v comm název `sleep`. Pokud spustíme binární soubor `~/hacking/sleep`, bude jeho jméno opět `sleep`. To je ovšem nepřijatelné. Proto je správnou volbou získat absolutní cestu ke spustitelnému souboru, jehož spuštěním vznikl proces s PID, které jsme zjistili. Absolutní cesta se získá přečtením symbolického odkazu `/proc/[pid]/exe`. [27]

Proto blacklisty a whitelisty, které jsou založené na jménech programů, musí uvádět absolutní cestu k binárnímu souboru, jehož spuštěním vzniká proces, kterému chtějí povolit/zakázat možnost získání dat. Pokud by nebyla ve whitelistu uvedená absolutní cesta, měli by automaticky schválený přístup veškeré procesy, jejichž binární soubor se jmenuje stejně jako jméno uvedené ve whitelistu. To by byl velký problém, neboť k autentizaci by stačilo pouze přejmenovat binární soubor. Do whitelistu by ovšem měly být umisťovány programy z rozumných míst jako je: `/usr/bin`, `/bin`, `/sbin` a dalších míst, do kterých má právo zápisu pouze administrátor. Jinak by mohlo dojít k nahrazení binárního souboru aplikace, která je ve whitelistu oprávněně, na jakýkoliv jiný, který by již ve whitelistu být neměl.

U blacklistů by názvy bez absolutní cesty takový problém nebyl (pokud by se název odvodil

z `/proc/[pid]/exe`). Blacklist by pouze přísnější. Vadilo by to pouze z toho hlediska, že by některé aplikace byly odmítnuté pouze na základě toho, že se shodují ve jméně s nějakou zakázanou aplikací, protože by nebylo zohledněno jejich umístění. Blacklist má spíše takovou funkci, že do něj přidáme například clipboard manager, aby nám stále od něj nechodily požadavky na data. Nicméně je možné ho obejít změnou názvu programu. Bylo by možné implementovat sofistikovanější implementaci, která by odmítla všechny procesy, které nemají binární soubory umístěny v důvěryhodných adresářích, kam má přístup k zápisu pouze root. Nicméně může se stát, že se něco v blacklistu zanedbá, a tudíž je tedy dobré se na něj plně nespoléhat. U každého procesu, který není nalezen ani v blacklistu ani ve whitelistu, je nutností se explicitně dotázat, zda uživatel souhlasí s poskytnutím dat tomuto procesu.

Velmi zásadní je také umístění těchto ACL. Musí být zajištěno, aby nebylo možné je jednoduše modifikovat. Proto je důležité kdo je vlastníkem těchto souborů a jaká mají práva. Vlastíkem i vlastnickou skupinou musí být root. Je-li tomu tak, nemůže pak práva souborů a ani vlastnictví měnit neprivilegovaný proces. Práva by měla zajistit, aby ostatní uživatelé mohly z těchto ACL číst, ale neměli možnost je modifikovat. Čili se nabízí nastavit práva a vlastnictví takto: `-rw-r--r-- root:root`. Důležitý je ovšem i adresář, ve které jsou ACL uložené. Protože jsou-li všechny předešlé podmínky z hlediska vlastnictví a práv splněny, ale soubor je uložen v adresáři, jehož vlastníkem je neprivilegovaný uživatel, může nastat problém. Uživatel může soubory s ACL smazat, pokud jsou umístěny v adresáři, který vlastní onen neprivilegovaný uživatel. Respektive neprivilegovaný uživatel nemůže ACL smazat, neboť nemá právo na jejich modifikaci, ale může smazat odkaz na tento soubor ze svého adresáře. Pokud nebude existovat nějaký jiný pevný odkaz na tento soubor, soubor je smazán filesystémem. Tím tedy je docíleno toho, že se vyřadí blacklisty i whitelisty. Je tedy nutné umisťovat blacklisty/whitelisty do adresáře, jako je například `/etc`.

5.2.3 Fungování nástroje pro bezpečné kopírování

Ovládání nástroje pro bezpečné kopírování, který byl implenetován v rámci této práce, je jednoduché. Stačí program spustit a v parametru mu předat data, která chceme bezpečně předat. Je zde možné pomocí přepínačů `-b` a `-w` dodat cestu k blacklistu/whitelistu. Pokud jsme již data zkopírovali tímto nástrojem, čeká se na operaci vložení. Po jejím vyvolání je provedena kontrola blacklistu (pokud existuje). V případu nalezení shody je předání dat zamítnuto. Jinak se pokračuje kontrolou whitelistu. V případě shody jsou data okamžitě předána. V opačném případě jsou vypsány informace o protistraně, která má zájem o naše data. Konkrétně se jedná o PID, rouru, FD a název protistrany. Jsme dotázáni, jestli chceme poskytnout data této protistraně. Po odpovědi YES jsou data zaslána. V případě jiné odpovědi nejsou zaslána.

Formát blacklistu a whitelistu je takový, že blacklist/whitelist musí obsahovat absolutní cesty ke spustitelným souborům oddělené novým řádkem. V případě nedodržení tohoto formátu končí program chybou. V případě špatného nastavení vlastníka a přístupových práv k blacklistu/whitelistu je tato skutečnost detekována a program je ukončen s chybou.

Každý program, který chce vložit data, je evidován v logu. Logováno je úspěšné i neúspěšné předání dat, a také podezřelé chování aplikace. Jako podezřelé chování aplikace se považuje, pokud najdeme více FD odkazujících na čtecí konec roury.

5.2.4 Zakomponování řešení do správce hesel

Nyní si ukážeme, jakým způsobem je možné zakomponovat nástroj pro bezpečné kopírování do správce hesel. Správce hesel, který byl k tomuto účelu vybrán, je Pass. Již jsme si ukazovali, jakým způsobem vypadá funkce tohoto správce hesel, která zajišťuje kopírování hesla do clipboardu. Nyní si ukážeme předefinovanou funkci, která podporuje pouze Wayland a používá náš nástroj pro bezpečné kopírování. Pomocí znaku `#` jsou označený řádky staré funkce, které nyní můžeme vypustit. Pomocí znaku `-->` jsou označeny řádky, které byly pozměněny.

Výpis kódu 5.6 Bezpečnější funkce programu Pass. [17]

```
clip() {
-->if [[ -n $WAYLAND_DISPLAY ]] && command -v secure-copy &> /dev/null;
        then
-->     local copy_cmd=( secure-copy \
                        -b /path/to/blacklist -w /path/to/whitelist )
                #local paste_cmd=( wl-paste -n )
                #if [[ $X_SELECTION == primary ]];
                #then
                #       copy_cmd+=( --primary )
                #       paste_cmd+=( --primary )
                #fi
                local display_name="$WAYLAND_DISPLAY"
        #elif [[ -n $DISPLAY ]] && command -v xclip &> /dev/null;
        #then
   #    local copy_cmd=( xclip -selection "$X_SELECTION" )
        #       local paste_cmd=( xclip -o -selection "$X_SELECTION" )
        #       local display_name="$DISPLAY"
        else
-->     die "Error: No Wayland display and clipper detected"
        fi
        #local sleep_argv0="password store sleep on display $display_name"

        #pkill -f "^$sleep_argv0" 2>/dev/null && sleep 0.5
        #local before="$("${paste_cmd[@]}" 2>/dev/null | $BASE64)"
--> "${copy_cmd[@]}" "$1" || die \
                  "Error: Could not copy data to the clipboard"
        #(
        #       ( exec -a "$sleep_argv0" bash <<<"trap 'kill %1' TERM; \
        #         sleep '$CLIP_TIME' & wait" )
        #       local now="$("${paste_cmd[@]}" | $BASE64)"
        #       [[ $now != $(echo -n "$1" | $BASE64) ]] && before="$now"
        #       qdbus org.kde.klipper /klipper \
        #         org.kde.klipper.klipper.clearClipboardHistory &>/dev/null
        #       echo "$before" | $BASE64 -d | "${copy_cmd[@]}"
        #) >/dev/null 2>&1 & disown
-->     echo "Copied $2 to clipboard." #Will clear in $CLIP_TIME seconds."
}
```

Jak vidíme, většinu řádků je nyní možné vypustit. Díky našemu nástroji se heslo neuloží do clipboard manageru (pokud to uživatel nechce), tudíž není potřeba clipboard/clipboard manager čistit. Heslo se přenese P2P z našeho nástroje do cílové aplikace. Po ukončení naší aplikace jsou zničeny všechny její objekty, tudíž je zničen i objekt `wl_data_source`.

5.2.5 Silné a slabé stránky řešení/implementace

Silná stránka tohoto řešení je především jednoduchost provedení a nasazení. Vzhledem k tomu, že není nutné měnit protokol ani implementaci na straně kompozitoru, je toto řešení prakticky okamžitě nasaditelné na libovolném Linuxovém OS s Waylandovým clipboardem. Řešení efektivně snižuje pravděpodobnost úniku tajné informace, která je předávána implementovaným nástrojem v rámci této práce. Tento nástroj je možné zakomponovat do správce hesel.

Další silnou stránkou tohoto řešení, která je implementována v nástroji pro bezpečné kopírování, je logování. Nástroj loguje, komu byla předána data, komu je odmítl předat a z jakého důvodu.

Slabou stránkou tohoto řešení je, že bezpečné kopírování je umožněno pouze aplikacím, které

se pro implementaci tohoto řešení rozhodnou. Pro dosažení bezpečného kopírování u všech aplikací je nutné, aby v každé z nich bylo implementováno toto řešení.

Další slabou stránkou tohoto řešení je, že v případě špatné práce s rourou u klienta, který chce přijmout data, může dojít k únikům dat. Pokud po vytvoření roury a předání FD nezačne klient okamžitě číst z FD (což je požadováno v protokolu), může se stát, že se zapíší data do roury dříve, než z ní kdokoliv začne číst. Následně je zde časové okno, které umožní, aby si FD nějaký proces duplikoval a začal z něj číst jako první. Rouru ovšem vytváří klient, který přijímá data, tudíž je v jeho zóně odpovědnosti, a proto špatná práce s ní není chybou tohoto řešení nýbrž chybou aplikace přijímající data.

Problematické je také to, že některé aplikace umisťují na FD timeout. Proto je možné jim předat data pouze tehdy, když se nachází ve whitelistu. Čekání na potvrzení od uživatele je moc dlouhá doba, a tudíž dojde vlivem timeoutu k uzavření FD čtecího konce roury.

Poslední ze slabých stránek je, že implementovaný nástroj kontroluje práva k souboru blacklistu/whitelistu pouze v základní formě. Pokud jsou použity rozšiřující POSIX ACL, není to rozpoznáno. Nicméně bylo by možné podporu pro ACL doimplementovat.

Závěr

Smyslem této bakalářské práce bylo zmapovat problematiku clipboardu, který je místem potencionálních úniků citlivých dat. V rámci této práce se podařilo získat reálný obraz bezpečnostní situace, v jaké se clipboard obecně nachází. Poměrně velkým překvapením bylo, že bezpečnostní hrozba pastejacking je stále aktuální a poměrně vážná.

Díky podrobnější analýze Linuxových clipboardů, zvláště Wayladnového clipboardu, bylo možné hlouběji porozumět fungování tohoto nástroje. Protokoly na předávání dat byly poměrně složité, což bylo poměrně překvapivé zjištění vzhledem k jednoduchosti tohoto nástroje. Vzhledem k tomu, že Waylandový clipboard je poměrně nový, chyběla dokumentace v podobě příkladů použití tohoto nástroje, což značně komplikovalo pochopení principů, na kterých clipboard funguje.

Specifické hrozby Linuxových implementací byly zmapovány a ohodnoceny metodikou CVSS. V reakci na tyto hrozby byla navržena řešení. Jejich silné a slabé stránky byly diskutovány. Těžištěm těchto řešení je Waylandový clipboard, neboť se jedná o implementaci, která je nástupcem X.Org, a proto byla vyhodnocena jako prioritní.

Po následném pochopení fungování clipboardu bylo možné implementovat nástroj, který je schopný bezpečným způsobem předat kopírovaná data. Nástroj je plně funkční a plně zdokumentovaný, tudíž je možné ho zakomponovat do správce hesel. Možnost zakomponování do správce hesel byla ukázána v rámci této práce. Tento nástroj umožňuje předávat data P2P a vyhnout se clipboard manageru. Díky logování, které tento nástroj využívá, je možné dohledat, které aplikace požadovaly vložení hesel a zda jim heslo bylo poskytnuto nebo ne.

Kromě nástroje pro bezpečné kopírování zahrnuje práce další programy, které slouží k vysvětlení problematiky, nebo dokazují vyzkoumanou zkutečnost. Jedním z nich je program, který vkládá data z Waylandového clipboardu pomocí Wlr_data_control protokolu.

Příloha A

Unix domain socket

Výpis kódu A.1 Unix domain socket server. [26]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#define SOCKET_NAME "/home/benjamin/somesocket.sock"

void clean(int data_socket, int connection_socket) {
  close(data_socket);
  close(connection_socket);
  unlink(SOCKET_NAME);
}

int main(int argc, char *argv[]) {
  struct sockaddr_un name;
  int ret, connection_socket, data_socket;
  char buffer[12];
  /*
   * for SO_PASSCRED begin */
  struct msghdr msgh;
  msgh.msg_name = NULL;
  msgh.msg_namelen = 0;
  struct iovec iov;
  int data;
  msgh.msg_iov = &iov;
  msgh.msg_iovlen = 1;
  iov.iov_base = &data;
  iov.iov_len = sizeof(data);
  union {
    char buf[CMSG_SPACE(sizeof(struct ucred))];
    struct cmsghdr align;
  } controlMsg;
  msgh.msg_control = controlMsg.buf;
  msgh.msg_controllen = sizeof(controlMsg.buf);
  struct cmsghdr *cmsgp;
  /*
   * for SO_PASSCRED end */
```

```
  struct ucred rcred, cred_peer;
  socklen_t cred_peer_len = sizeof(cred_peer);
  connection_socket = socket(AF_UNIX, SOCK_STREAM, 0);

  if (connection_socket == -1) {
    perror("socket");
    exit(EXIT_FAILURE);
  }

  memset(&name, 0, sizeof(name));
  name.sun_family = AF_UNIX;
  strncpy(name.sun_path, SOCKET_NAME, sizeof(name.sun_path) - 1);

  if (bind(connection_socket, (const struct sockaddr *)&name,
           sizeof(name))==-1) {
    close(connection_socket);
    unlink(SOCKET_NAME);
    exit(EXIT_FAILURE);
  }

  if (listen(connection_socket, 8) == -1) {
    close(connection_socket);
    unlink(SOCKET_NAME);
    exit(EXIT_FAILURE);
  }
  data_socket = accept(connection_socket, NULL, NULL);

  if (data_socket == -1) {
    close(connection_socket);
    unlink(SOCKET_NAME);
    exit(EXIT_FAILURE);
  }

  int optval = 1;

  if (setsockopt(data_socket, SOL_SOCKET, SO_PASSCRED, &optval,
                 sizeof(optval)) == -1) {
    clean(data_socket, connection_socket);
    exit(EXIT_FAILURE);
  }


/* SO_PASSCRED begin */
  recvmsg(data_socket, &msgh, 0);
  cmsgp = CMSG_FIRSTHDR(&msgh);
  memset(&rcred, 0, sizeof(rcred));
  if (cmsgp->cmsg_level != SOL_SOCKET
      || cmsgp->cmsg_type != SCM_CREDENTIALS) {
    clean(data_socket, connection_socket);
    exit(EXIT_FAILURE);
  }
  memcpy(&rcred, CMSG_DATA(cmsgp), sizeof(struct ucred));
  printf("SO_PASSCRED pid=%ld, uid=%ld, gid=%ld\n",
         (long)rcred.pid, (long)rcred.uid, (long)rcred.gid);
/* SO_PASSCRED end */
/* SO_PEERCRED begin */
  memset(&cred_peer, 0, sizeof(cred_peer));
```

```
  getsockopt(data_socket, SOL_SOCKET, SO_PEERCRED,
             &cred_peer, &cred_peer_len);
  printf("SO_PEERCRED pid=%ld, uid=%ld, gid=%ld\n",
         (long)cred_peer.pid,(long)cred_peer.uid, (long)cred_peer.gid);
/* SO_PEERCRED end */
/* reading before fork() */
  if (read(data_socket, buffer, sizeof(buffer)) == -1) {
    clean(data_socket, connection_socket);
    exit(EXIT_FAILURE);
  }

/*SO_PASSCRED begin */
  recvmsg(data_socket, &msgh, 0);
  cmsgp = CMSG_FIRSTHDR(&msgh);
  memset(&rcred, 0, sizeof(rcred));
  if (cmsgp->cmsg_level != SOL_SOCKET
      || cmsgp->cmsg_type != SCM_CREDENTIALS) {
    clean(data_socket, connection_socket);
    exit(EXIT_FAILURE);
  }
  memcpy(&rcred, CMSG_DATA(cmsgp), sizeof(struct ucred));
  printf("SO_PASSCRED after fork() pid=%ld, uid=%ld, gid=%ld\n",
         (long)rcred.pid, (long)rcred.uid, (long)rcred.gid);

/* SO_PASSCRED end */
/* SO_PEERCRED begin */
  memset(&cred_peer, 0, sizeof(cred_peer));
  getsockopt(data_socket, SOL_SOCKET, SO_PEERCRED,
             &cred_peer, &cred_peer_len);
  printf("SO_PEERCRED after fork() pid=%ld, uid=%ld, gid=%ld\n",
         (long)cred_peer.pid, (long)cred_peer.uid, (long)cred_peer.gid);
/* SO_PEERCRED end */
/* reading after fork() */
  ret = read(data_socket, buffer, sizeof(buffer));

  if (ret == -1) {
    clean(data_socket, connection_socket);
    exit(EXIT_FAILURE);
  }

  clean(data_socket, connection_socket);
  exit(EXIT_SUCCESS);
}
```

```
$ ./server.out
SO_PASSCRED pid=8059, uid=1000, gid=1000
SO_PEERCRED pid=8059, uid=1000, gid=1000
SO_PASSCRED after fork() pid=8060, uid=1000, gid=1000
SO_PEERCRED after fork() pid=8059, uid=1000, gid=1000
```

Výpis kódu A.2 Unix domain socket client. [26]

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <sys/wait.h>
#define SOCKET_NAME "/home/benjamin/somesocket.sock"

int main(int argc, char *argv[]) {
  struct sockaddr_un addr;
  int ret;
  int data_socket;

  data_socket = socket(AF_UNIX, SOCK_STREAM, 0);
  if (data_socket == -1) {
    perror("socket");
    exit(EXIT_FAILURE);
  }
  memset(&addr, 0, sizeof(addr));

  addr.sun_family = AF_UNIX;
  strncpy(addr.sun_path, SOCKET_NAME, sizeof(addr.sun_path) - 1);

  ret = connect(data_socket, (const struct sockaddr *)&addr, sizeof(addr));
  if (ret == -1) {
    fprintf(stderr, "The server is down.\n");
    exit(EXIT_FAILURE);
  }

  ret = write(data_socket, "123456789", 10);
  pid_t fork_ret = fork();
  if (fork_ret < 0)
    return -1;
  else if (fork_ret == 0) {
    /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */
    ret = write(data_socket, "123456789", 10);
    if (ret == -1) {
      perror("write");
      exit(EXIT_FAILURE);
    }
    return 0;
  } else {
    int dummy;
    waitpid(fork_ret, &dummy, 0);

    close(data_socket);

    exit(EXIT_SUCCESS);
  }
}
```

Příloha B

Vyhodnocení hrozeb

Tabulka B.1 Vyhodnocení hrozby pastejackingu – uživatelský terminál.

Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	Required
Scope	Changed
Confidentiality	Low
Integrity	Low
Availability	Low
Celkové skóre	7,1

Tabulka B.2 Vyhodnocení hrozby pastejackingu – administrátorský terminál.

Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	Required
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High
Celkové skóre	9,6

Tabulka B.3 Vyhodnocení hrozeb X.Org clipboardu – spuštění škodlivého příkazu.

Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	Required
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High
Celkové skóre	8,2

Tabulka B.4 Vyhodnocení hrozeb X.Org clipboardu – únik tajných informací.

Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	None
Availability	None
Celkové skóre	5

Tabulka B.5 Vyhodnocení hrozeb clipboardu OS Android – únik tajných informací.

Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	None
Availability	None
Celkové skóre	5

Tabulka B.6 Vyhodnocení hrozeb Waylandového clipboardu – spuštění škodlivého příkazu.

Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	Required
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High
Celkové skóre	8,2

Tabulka B.7 Vyhodnocení hrozeb Waylandového clipboardu – únik tajných informací.

Attack Vector	Local
Attack Complexity	Low
Privileges Required	Low
User Interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	None
Availability	None
Celkové skóre	5

Bibliografie

1. ROSENTHAL, David. *Inter-Client Communication Conventions Manual* [online]. Ve spol. s MARKS, Stuart W. [cit. 2022-05-04]. Dostupné z: `https://www.x.org/releases/current/doc/xorg-docs/icccm/icccm.html#Peer_to_Peer_Communication_by_Means_of_Selections`.
2. MOOLENAAR, Bram. *VIM USER MANUAL* [online]. 2020 [cit. 2022-05-04]. Dostupné z: `https://vimhelp.org/usr_toc.txt.html`.
3. MICROSOFT CORPORATION. *Clipboard Formats - Win32 apps* [online]. In collab. with ALVINASHCRAFT; DCTHEGEEK; METATHINKER; CHRIS-E-WATSON; JBCOOK; MIJACOBS; MSATRANJR [cit. 2022-05-04]. Dostupné z: `https://docs.microsoft.com/en-us/windows/win32/dataxchg/clipboard-formats`.
4. APPLE INC. *UIPasteboard | Apple Developer Documentation* [online] [cit. 2022-05-07]. Dostupné z: `https://developer.apple.com/documentation/uikit/uipasteboard`.
5. HØGSBERG, Kristian. *Wayland* [The Wayland Protocol] [online]. 2012 [cit. 2022-05-07]. Dostupné z: `https://wayland.freedesktop.org/docs/html/index.html`.
6. ALVINASHCRAFT. *Clipboard Class (Windows.ApplicationModel.DataTransfer) - Windows UWP applications* [online] [cit. 2022-05-07]. Dostupné z: `https://docs.microsoft.com/en-us/uwp/api/windows.applicationmodel.datatransfer.clipboard`.
7. RODRIGUES, Philip. *The Klipper Handbook* [online]. 2021 [cit. 2022-05-07]. Dostupné z: `https://docs.kde.org/stable5/en/plasma-workspace/klipper/index.html`.
8. *What is Pastejacking?* [GeeksforGeeks] [online]. 2020-08-10 [cit. 2022-05-07]. Dostupné z: `https://www.geeksforgeeks.org/what-is-pastejacking/`. Section: Computer Subject.
9. MDN CONTRIBUTORS. *Clipboard API - Web APIs | MDN* [online]. 2022-04-19 [cit. 2022-05-07]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/API/Clipboard_API`.
10. KERRISK, Michael. *readline(3) - Linux manual page* [online]. 2020-10-29 [cit. 2022-05-11]. Dostupné z: `https://www.man7.org/linux/man-pages/man3/readline.3.html`.
11. APPLE INC. *NSPasteboard | Apple Developer Documentation* [online] [cit. 2022-05-07]. Dostupné z: `https://developer.apple.com/documentation/appkit/nspasteboard`.
12. APPLE INC. *Pasteboard* [online] [cit. 2022-05-07]. Dostupné z: `https://1url.cz/BrzON`.
13. APPLE INC. *pbcopy(1) [osx man page]* [online]. 2005-01-12 [cit. 2022-05-07]. Dostupné z: `https://www.unix.com/man-page/osx/1/pbcopy/`.
14. SDWHEELER. *Get-Clipboard (Microsoft.PowerShell.Management) - PowerShell* [online] [cit. 2022-05-07]. Dostupné z: `https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-clipboard`.

15. SDWHEELER. *Set-Clipboard (Microsoft.PowerShell.Management) - PowerShell* [online] [cit. 2022-05-07]. Dostupné z: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/set-clipboard.
16. GOOGLE LLC. *Copy and Paste* [Android Developers] [online] [cit. 2022-05-07]. Dostupné z: https://developer.android.com/guide/topics/text/copy-paste.
17. DONENFELD, Jason A. *pass* [online] [cit. 2022-05-07]. Dostupné z: https://git.zx2c4.com/password-store/tree/src/password-store.sh.
18. REICHL, Dominik. *Auto-Type - KeePass* [online] [cit. 2022-05-07]. Dostupné z: https://keepass.info/help/base/autotype.html.
19. GETTYS, James; SCHEIFLER, Robert W.; ADAMS, Chuck; JOLOBOFF, Vania; HIURA, Hideki; MCMAHON, Bill; NEWMAN, Ron; TABAYOYON, Al; WIDENER, Glenn; YAMADA, Shigeru. *Xlib - C Language X Interface* [online] [cit. 2022-05-07]. Dostupné z: https://www.x.org/releases/current/doc/libX11/libX11/libX11.html.
20. MDN CONTRIBUTORS. *MIME types (IANA media types) - HTTP | MDN* [online]. 2022-05-01 [cit. 2022-05-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types.
21. ALLEN, Steve. *Internet Standards regarding MIME* [online]. 2003-05-18 [cit. 2022-05-08]. Dostupné z: https://www.ucolick.org/~sla/fits/mime/inetstds.html.
22. *Wayland Protocol Documentation | Wayland Explorer* [A better way to read Wayland documentation] [online]. 2021 [cit. 2022-05-07]. Dostupné z: https://wayland.app/protocols/.
23. SER, Simon. *Wayland clipboard and drag & drop · emersion* [online]. 2020-03-26 [cit. 2022-05-07]. Dostupné z: https://emersion.fr/blog/2020/wayland-clipboard-drag-and-drop/.
24. SAUNDERS, Kim; Ã…STRAND, Peter. *xclip(1): CLI to X selections - Linux man page* [online] [cit. 2022-05-08]. Dostupné z: https://linux.die.net/man/1/xclip.
25. CARPENTER, Preston; DEVAULT, Drew. *Introduction - The Wayland Protocol* [online] [cit. 2022-05-07]. Dostupné z: https://wayland-book.com/.
26. KERRISK, Michael. *unix(7) - Linux manual page* [online]. 2021-03-22 [cit. 2022-05-07]. Dostupné z: https://man7.org/linux/man-pages/man7/unix.7.html.
27. KERRISK, Michael. *proc(5) - Linux manual page* [online]. 2021-08-27 [cit. 2022-05-07]. Dostupné z: https://man7.org/linux/man-pages/man5/proc.5.html.
28. KERRISK, Michael. *pidfd_getfd(2) - Linux manual page* [online] [cit. 2022-05-08]. Dostupné z: https://man7.org/linux/man-pages/man2/pidfd_getfd.2.html.

Obsah přiloženého média

- `readme.txt` ... stručný popis obsahu média
- `impl` ... implementace včetně zdrojových kódů
  - `secure-copy` ... nástroj pro bezpečné kopírování
    - `build`
      - `secure-copy` ... spustitelný kód nástroje pro bezpečné kopírování
    - `doc` ... dokumentace nástroje pro bezpečné kopírování
  - `wl_paste` ... nástroj pro vkládání pomocí Wl_data protokolu
    - `build`
      - `wl-data-paste` . spustitelný kód nástroje pro vkládání pomocí Wl_data protokolu
  - `wlr_data_control_paste` .... nástroj pro vkládání pomocí Wlr_data_control protokolu
    - `build`
      - `wlr-paste` ..... spustitelný kód nástroje pro vkládání pomocí Wlr_data_control protokolu
- `text-prace` ... text práce včetně zdrojové formy ve formátu LaTeX
  - `build`
    - `ctufit-thesis.pdf` ... text práce ve formátu PDF