FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Útok Shatter a technologie User Interface Privilege Isolation |
| **Student:** | Adam Škoda |
| **Vedoucí:** | Ing. Josef Kokeš, Ph.D. |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Informační bezpečnost 2021 |
| **Katedra:** | Katedra informační bezpečnosti |
| **Platnost zadání:** | do konce letního semestru 2024/2025 |

## Pokyny pro vypracování

1) Nastudujte si útok Shatter ( https://en.wikipedia.org/wiki/Shatter_attack ) a popište principy, na kterých byl založen.
2) Proveďte rešerši úprav provedených Microsoftem v reakci na tyto útoky: Integrity Levels (IL), User Interface Privilege Isolation (UIPI).
3) Vytvořte demonstrační aplikaci, na které ukážete:
a) chování systému v jeho základním nastavení,
b) jak se aplikace může k využití těchto technologií přihlásit, a
c) jak se změní chování aplikace s těmito technologiemi aktivovanými (např. proveďte experimenty s komunikací aplikací na různých IL a vyhodnoťte rozdíly proti standardnímu chování).
4) Diskutujte svá zjištění, formulujte doporučení pro vývojáře aplikací.

---

*Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 10. ledna 2024 v Praze.*

Bakalářská práce

# ÚTOK SHATTER A TECHNOLOGIE USER INTERFACE PRIVILEGE ISOLATION

**Adam Škoda**

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedoucí: Ing. Josef Kokeš, Ph.D.
8. května 2024

Odkaz na tuto práci: Škoda Adam. *Útok Shatter a technologie User Interface Privilege Isolation*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

# Obsah

# Seznam obrázků



# Seznam tabulek

# Seznam výpisů kódu

*Rád bych poděkoval Ing. Josefu Kokešovi, Ph.D. za četné rady a čas, který věnoval vedení této práce. Děkuji také svým blízkým, kteří mě podporují při 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 a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 8. května 2024

# Abstrakt

Tato práce rozebírá útoky typu *Shatter* a bezpečnostní prvky zavedené v reakci na něj. Zaměřuje se na technologie, na kterých je jak útok, tak obrana proti němu založena. Od teoretické rešerše následně přechází k experimentům, jejichž cílem je obrany analyzovat a porovnat jejich dokumentované chování s chováním skutečným. Při testování využívá práce řadu nově vzniknuvších aplikací vytvořených přímo za tímto účelem. Výsledky experimentů jsou poté zasazeny do širšího kontextu a hodnoceny z hlediska možných bezpečnostních nedostatků. Provedeným výzkumem byly zjištěny inkonzistence v chování některých ochranných prvků. Text práce se věnuje jejich hlubšímu rozboru a zkoumá možné metody zneužití. V závěru jsou formulována doporučení pro vývojáře aplikací, která z výsledků výzkumu vychází.

**Klíčová slova** Shatter, útok, Windows, window messages, integrity level, User Interface Privilege Isolation, elevace oprávnění, testovací program

# Abstract

This thesis analyzes *Shatter*-style attacks and the security mechanisms introduced to mitigate them. Its focus lies on the technologies that the attack and the defenses against it are based on. After laying down the necessary theoretical groundwork, it transitions to experimentation with the goal of analyzing the established defenses and comparing their actual behavior with the behavior that has been previously documented. To this end, several newly created programs are used. The results are put into a broader context and evaluated in terms of security deficiencies. There have been inconsistencies identified in the ways that certain defense mechanisms behave. The thesis then expands upon them and explores possible avenues of their exploitation. The last chapter leverages the results of the conducted experiments to formulate recommendations for software developers.

**Keywords** Shatter, attack, Windows, window messages, integrity level, User Interface Privilege Isolation, privilege escalation, testing program

# Seznam zkratek

| | |
|---|---|
| ACE | Access Control Entry |
| ACL | Access Control List |
| API | Application Programming Interface |
| ASLR | Address Space Layout Randomization |
| COM | Component Object Model |
| CTF | Common Text Framework |
| DACL | Discretionary Access Control List |
| DDE | Dynamic Data Exchange |
| DEP | Data Execution Prevention |
| DLL | Dynamic-Link Library |
| DoS | Denial of Service |
| DRM | Digital Rights Management |
| DWM | Desktop Window Manager |
| exFAT | Extensible File Allocation Table |
| FAT | File Allocation Table |
| GDI | Graphics Device Interface |
| GUI | Graphical User Interface |
| IL | Integrity Level |
| IME | Input Method Editor |
| IPC | Inter-Process Communication |
| MDI | Multiple-Document Interface |
| MIC | Mandatory Integrity Control |
| NTFS | New Technology File System |
| PDF | Portable Document Format |
| PEB | Process Environment Block |
| RID | Relative Identifier |
| RPC | Remote Procedure Call |
| SACL | System Access Control List |
| SDDL | Security Descriptor Definition Language |
| SEH | Structured Exception Handling |
| SID | Security Identifier |
| SO | Securable Object |
| SQL | Structured Query Language |
| SRM | Security Reference Monitor |
| TEB | Thread Environment Block |
| TSF | Text Services Framework |
| UAC | User Account Control |
| UIPI | User Interface Privilege Isolation |
| XML | Extensible Markup Language |

# Slovníček pojmů

| | |
|---|---|
| atom | 16bitový identifikátor používaný k přístupu do tabulky atomů, která obsahuje jména atomů v podobě řetězců [1] |
| *botnet* | skupina kompromitovaných strojů, které útočník vzdáleně ovládá [2] |
| *brute-force* útok | útok spočívající v systematickém prohledávání celého prostoru všech možných voleb (typicky je takto útočeno na hesla) |
| *buffer overflow* | situace, kdy je do paměťového *bufferu* zapsáno více dat, než je jeho kapacita, a dojde k přepsání dat ležících za *bufferem* [3] |
| *callback* | funkce, která je předána jako argument jiné funkci, v níž je následně volána [4] |
| *debugger* | program umožnující diagnostiku a odstraňování chyb v programech |
| *denial of service* útok | útok na server, službu nebo jiný sdílený prostředek s cílem jej učinit plně nebo částečně nedostupným pro legitimní uživatele [5] |
| *desktop object* | objekt reprezentující plochu ve *Windows* a obsahující *security descriptor* [6] |
| DLL *loader* | část operačního systému, která načítá DLL [7] |
| *exception handler* | funkce, která je spouštěna při výjimce |
| *exploit* | část programu nebo dat případně sekvence příkazů zneužívající zranitelnost [8] |
| *file inclusion* | situace, kdy útočník vlivem nedostatečné validace vstupu dokáže zobrazit, upravit či spustit soubor na cílovém stroji [9] |
| *full virtualization* | typ virtualizace, při kterém virtualizační software plně simuluje hardware [10] |
| *injection* | situace, kdy je uživatelský vstup vlivem jeho nedostatečné validace interpretován jako spustitelný kód [11] |
| *load* adresa | adresa, od které je program zaveden do paměti |
| *localhost* | rezervovaná IP adresa odkazující na aktuálně používaný stroj [12] |
| *logon session* | relace, která počíná přihlášením uživatele a končí jeho odhlášením [13] |
| *message loop* | smyčka ve funkci `wWinMain`, která voláním `GetMessage` a `DispatchMessage` opakovaně vybírá *window messages* z fronty a předává je cílovým oknům [14] |
| `NOP` | instrukce, která neprovede žádnou operaci [15] |
| *patch* | okamžité řešení identifikovaného problému, které je poskytováno uživatelům [16] |
| *payload* | část programu nebo dat vykonávající škodlivou operaci |
| *privilege escalation* | situace, kdy útočník zneužije zranitelnost a získá vyšší oprávnění než by mu byla přiřazena za normálních okolností [17, 18] |
| *reverse shell* | *shell*, který je navázán z cílového zařízení zpět na útočníkovo zařízení |
| *sandbox* | omezené prostředí, které omezuje přístup programů k systémovým prostředkům [19] |
| *shell* | program, který umožňuje uživateli přímo zadávat příkazy operačnímu systému a vystavuje jeho rozhraní [20] |

# Úvod

První útok typu *Shatter* byl proveden již v roce 2002 na operačním systému Windows 2000 Professional. Okolnosti tehdy nasvědčovaly tomu, že se jedná o zásadní bezpečnostní slabinu, kterou Microsoft nebude schopen jednoduše opravit. Několik let na to vznikl koncept *integrity levelů* (IL) společně s technologií *User Interface Privilege Isolation* (UIPI), se kterými se ve Windows setkáváme dodnes. Avšak i přes stáří těchto technologií je dokumentace, zejména u UIPI, poměrně chudá a vzniká tak potřeba ji doplnit.

Hodnota a prospěšnost tohoto tématu tkví zejména v jeho neprobádanosti, kde se protíná i s motivací jeho výběru. Poskytuje jak teoretický přehled pro ty, kteří se do něj chtějí ponořit hlouběji, tak doporučení cílená zejména na vývojáře aplikací, kde zdůrazňuje, jakým způsobem poznatky aplikovat tak, aby nově vznikající aplikace nebyly tímto typem útoku zranitelné.

Cílem práce je tedy zejména zdokumentovat a popsat útok *Shatter*, a to včetně mechanismů a principů, na kterých je založen. Stejným způsobem se práce snaží analyzovat technologie, které na obranu před těmito útoky vznikly – *User Interface Privilege Isolation* a *integrity levels*. Za cíl si práce klade i praktickou analýzu, která rešeršní část doplňuje a věnuje se porovnání chování aplikací napříč různými úrovněmi ochrany či zkoumá chování systému v jeho výchozím nastavení. Na základě těchto experimentů je cílem formulovat zmiňovaná doporučení pro vývojáře aplikací.

V teoretické části práce jsou postupně rozebírány veškeré technologie, které mají s útoky typu *Shatter* souvislost. Práce je uvedena kapitolou o *window messages*, jejíž cílem je čtenáře seznámit se základy, na nichž útoky typu *Shatter* stojí. Na ní navazuje samotný popis útoků typu *Shatter*, včetně rekonstrukce původního typu útoku a rozebrání mechanismů jeho fungování. Za cíl si tato kapitola klade popsat způsoby, kterými útoky fungují, vzájemně je porovnat a shrnout opatření, která v reakci na ně byla navržena.

Druhá část práce rozebírá bezpečnostní mechanismy, které byly v reakci na útoky typu *Shatter* zavedeny: *Mandatory Integrity Control* (MIC) a *User Inteface Privilege Isolation*. Obě technologie jsou nejprve uvedeny teorií a následně práce přechází do praxe v podobně experimentů. V rámci *Mandatory Integrity Control* je věnována pozornost zejména jednotlivým úrovním *integrity levelů* a mechanismům s nimi spjatých, jako je dědění či nastavování na souborech a procesech. Podobně jako u *Mandatory Integrity Control* si sekce o *User Inteface Privilege Isolation* klade za cíl provázat teorii s praxí a sleduje, jakým způsobem mohou aplikace vzájemně interagovat napříč různými *integrity levely*. Na základě poznatků plynoucích z experimentu jsou formulována doporučení pro vývojáře.

# Kapitola 1

# *Window messages*

Před tím, než bude přistoupeno k samotnému popisu útoků *Shatter*, je nutné porozumět technologii, která jim dala vzniknout – *window messages*[1].

Na rozdíl od předchůdců operačního systému Windows, jako byl např. MS-DOS, je grafické rozhraní Windows *event-driven*. To znamená, že namísto toho, aby se aplikace sama dotazovala na uživatelský vstup, je jí vstup automaticky předáván operačním systémem [21]. Ve Windows jsou tyto události doručovány ve formě *window messages*.

Každá zpráva upozorňuje aplikaci na to, že by na nastalou událost měla reagovat, a poskytuje informace nutné pro její zpracování. Windows používá systém předávání zpráv nejen pro zmiňovaný uživatelský vstup, jako např. posun kurzoru, kliknutí tlačítka myši nebo stisk klávesy, ale také pro systémové události. Mezi ně pak patří požadavek na vykreslení obsahu okna nebo informace o tom, že systém přechází do režimu hibernace, tedy události, které nemají ve fyzickém světě analogii [22]. Důležité je také zmínit, že aplikace mohou pomocí *window messages* komunikovat jak s okny, která jim přímo náleží, tak s okny jiných naprosto nezávislých aplikací. [21].

## 1.1 Zpracování

Typickým způsobem zpracování *window messages* je *message loop*. Zde je ve smyčce opakovaně volána funkce `GetMessage`, která zajistí vyjmutí zprávy z fronty zpráv a její předání do *window procedure* příslušného okna pomocí `DispatchMessage`. [21]

*Window procedure* je funkce, kterou by měla definovat každá aplikace zobrazující okna v grafickém rozhraní Windows. Tato funkce zodpovídá za zpracování veškerých zpráv, tedy událostí, které nastávají v oknech dané třídy. Není typické, že by některé třídy oken ignorovaly určité druhy zpráv, proto by v případě, že pro zpracování dané zprávy není potřeba provést žádnou akci, měla *window procedure* volat funkci `DefWindowProc`, která zajistí, že cílové okno zareaguje výchozím způsobem. [23]

*Window messages* můžeme rozdělit do dvou skupin, na *queued* a *non-queued messages*. *Queued messages* jsou zprávy, které jsou vkládány do fronty zpráv. To znamená, že jsou nejprve v *message loop* z fronty vyjmuty pomocí `GetMessage`, následně mohou být zpracovány a poté jsou pomocí `DispatchMessage` předány do *window procedure* patřičného okna, která zajistí správnou reakci [21]. *Non-queued messages* se od předešlého typu zpráv liší tím, že se v *message loop* neprojeví a jsou rovnou předávány příslušné *window procedure* [21]. Poněkud neintuitivní je, že pro jejich doručení musí být nejprve zavolána funkce `GetMessage`. Při jejím volání jsou doručeny všechny čekající *non-queued* zprávy, které se ale v `GetMessage` neprojeví, a až poté jsou vybírány *queued* zprávy [24]. K vybírání zpráv totiž dochází v následujícím pořadí [24]:

[1] Dále označované také jako „zprávy.“

1. *non-queued* zprávy (zprávy poslané pomocí `SendMessage`);
2. *queued* zprávy (zprávy poslané pomocí `PostMessage`);
3. zprávy pocházející od hardwarových vstupních zařízení;
4. opět *non-queued* zprávy;
5. zprávy `WM_PAINT`;
6. zprávy `WM_TIMER`.

Přestože Microsoft dokumentace nazývá některé zprávy výlučně *queued* (např. `WM_PAINT`) nebo *non-queued* (např. `WM_ACTIVATE`), neznamená to, že by *non-queued* zprávy nemohly být poslány pomocí `PostMessage` ve formě *queued* zpráv a naopak. Z tohoto pohledu se jedná o informativní dělení, které je dodržováno v případě, že původcem těchto zpráv je operační systém. Pokud jde o zprávy, které jsou generovány aplikacemi, nelze se na tuto klasifikaci spolehnout.

## 1.2 Obsah

Podoba, ve které aplikace obdrží *window message*, se odvíjí od fáze jejího zpracování. Z funkční signatury typické *window procedure* (ve výpisu kódu 1.1) stanovené C/C++ Windows *application programming interface* (API) vyplývá, že *window message* získává rozloženou na její jednotlivé části. [25]

**Výpis kódu 1.1** Funkční signatura Windows API *window procedure* [25]

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
        LPARAM lParam);
```

Z výpisu kódu 1.1 je zřejmé, že *window procedure* má v základu k dispozici atributy zprávy popsané v tabulce 1.1.

**Tabulka 1.1** Parametry Windows API *window procedure* [25]

| Typ parametru | Identifikátor | Význam |
|---|---|---|
| `HWND` | `hwnd` | *handle* okna |
| `UINT` | `uMsg` | číslo identifikující typ příchozí zprávy |
| `WPARAM` | `wParam` | první parametr zprávy |
| `LPARAM` | `lParam` | druhý parametr zprávy |

*Handle* `hwnd` slouží k identifikaci okna, kterého se zpráva týká. Parametr `uMsg` nese informaci o tom, o jakou zprávu se jedná, a tedy o tom, jak by na ni aplikace měla reagovat. Tato čísla jsou definována v celé řadě hlavičkových souborů Windows API, kde jim jsou přiřazeny srozumitelnější názvy, přičemž ty nejčastější z nich lze běžně najít ve `WinUser.h`. Zbývající dva parametry `wParam` a `lParam` fungují jako parametry samotné zprávy, ukládají tedy data nutná pro její správné zpracování. V případě, že nejsou využity, nastavují se typicky na hodnotu `NULL`. [25]

Mapování jednotlivých konstant identifikujících *window messages* na srozumitelnější jména lze najít ve zdrojovém kódu aplikací `sender.exe` a `receiver.exe`, které vznikly jako součást této bakalářské práce a jsou dostupné na přiloženém médiu. Seznam zpráv v tomto souboru sice jistě není vyčerpávající, ale obsahuje veškeré zprávy zmíněné v této práci.

**Výpis kódu 1.2** Obsah struktury `MSG` [26]

```
typedef struct tagMSG {
        HWND        hwnd;
        UINT        message;
        WPARAM      wParam;
        LPARAM      lParam;
        DWORD       time;
        POINT       pt;
#ifdef _MAC
        DWORD       lPrivate;
#endif
} MSG, *PMSG, NEAR *NPMSG, FAR *LPMSG;
```

V *message loop*, která obdrží zprávy odeslané pomocí `PostMessage` ještě před *window procedure*, jsou zprávy reprezentovány strukturou `MSG` (ve výpisu kódu 1.2), která je jedním ze vstupně výstupních parametrů funkce `GetMessage`. [26]

Atribut `message` struktury `MSG` označuje stejnou informaci jako parametr `uMsg` známý ze signatury *window procedure* (ve výpisu kódu 1.1). Kromě atributů reprezentujících parametry *window message* obsahuje `MSG` navíc další dvě položky, a to `time` značící čas odeslání zprávy a `pt`, který uchovává informaci o poloze kurzoru v momentu, kdy byla zpráva odeslána [26]. Atributy `time` resp. `pt` lze v případě potřeby z *window procedure* získat voláním funkcí `GetMessageTime` resp. `GetMessagePos` [21].

Atribut `lPrivate`, který je obalen podmíněným překladem na základě `_MAC`, je pozůstatek z dob, kdy byl Microsoft největším vývojářem software pro Macintosh. Na Windows se tento atribut ve *window messages* nepoužívá. [27]

## 1.3 Dělení

K vymezení zpráv vyhrazených pro operační systém a zpráv, které mohou aplikace volně používat, slouží konstanta `WM_APP` (`0x8000`) definovaná ve `WinUser.h` [28]. Ačkoliv by její pojmenování mohlo vést k závěru, že se jedná o typ *window message*, není tomu tak. Podobně jako `WM_USER` [29] (`0x400`) se používá pouze pro stanovení rozsahu zpráv s určitým využitím. Protože jsou ale tyto rozsahy inkluzivní, může se stát, že třída resp. aplikace použijí hodnotu `0x400` resp. `0x8000` pro definici vlastní *window message*. Konstanty `WM_USER` a `WM_APP` však samy o sobě žádný význam nemají. Dle dokumentace se *window messages* dělí do skupin podle tabulky 1.2.

**Tabulka 1.2** Rozsahy *window messages* a jejich významy [28, 29]

| Rozsah | Význam |
|---|---|
| `0x0` až `0x3ff` | zprávy rezervované pro operační systém |
| `0x400` až `0x7fff` | zprávy pro soukromé třídy oken |
| `0x8000` až `0xbfff` | zprávy pro použití aplikacemi |
| `0xc000` až `0xffff` | řetězcové zprávy pro použití aplikacemi[2] |
| výše než `0xffff` | zprávy rezervované pro operační systém |

Zprávy pro soukromé třídy oken jsou zprávy, které si mezi sebou mohou vyměňovat instance uživatelem definovaných tříd oken. Tyto zprávy by neměly být posílány napříč aplikací, protože

[2] V originále označovány „*string messages for use by applications.*“

mohou kolidovat se zprávami, které používají předdefinované typy oken, jako např. ovládací prvky *list box* nebo *button*. K tomuto účelu jsou určeny zprávy z rozsahu `WM_APP` až `0xbfff`. [28]

Rozsah `0xc000` až `0xffff` je používán pro zprávy, které byly registrovány za běhu aplikace pomocí funkce `RegisterWindowMessage` [28]. Pomocí této funkce může aplikace pro řetězec získat identifikátor zprávy a v případě, že několik aplikací použije stejný řetězec, mohou tento identifikátor využívat pro výměnu zpráv [28]. Identifikátor vracený `RegisterWindowMessage` koresponduje s parametrem `uMsg` resp. atributem `message` z výpisu kódu 1.1 resp. 1.2. Mezi tyto *window messages* patří i zprávy *Common Text Frameworku* (CTF) [30] využívající řetězce `MSUIM.Msg.LangBarModal` a `MSUIM.Msg.Private`, které jsou do detailu rozebírány v sekci 4.4.6.8.

## 1.4 Odesílání

Pokud chce aplikace poslat *window message*, nabízejí se dvě možnosti. Může použít funkci `PostMessage` k odeslání *queued message*, tedy zprávy, která bude zařazena do fronty zpráv a nejprve zpracována v *message loop*. Alternativou je funkce `SendMessage`, která vytvoří *non-queued message*, tedy zprávu, která je rovnou předána *window procedure* cílového okna. Druhý přístup se používá, pokud je žádoucí, aby okno reagovalo na událost ihned. Existuje ještě několik dalších funkcí jako `SendMessageCallback` nebo `BroadcastSystemMessage`, které rozšiřují `PostMessage` a `SendMessage`, nicméně princip jejich fungování je téměř totožný. [21]

Funkce `PostMessage` a `SendMessage` jsou až na několik důležitých rozdílů, jako je práce s frontou zpráv, téměř totožné. Prvním takovým rozdílem je, že volání `SendMessage` nebude dokončeno, dokud cílová *window procedure* nezpracuje předanou zprávu [31], kdežto `PostMessage` se z volání vrací ihned [32]. Druhým podstatným rozdílem je návratová hodnota obou funkcí. Funkce `SendMessage` vrací typ `LRESULT` a dokumentace specifikuje, že návratová hodnota závisí na odeslané zprávě [31]. Naproti tomu návratová hodnota `PostMessage` má typ `BOOL`, který je nenulový při úspěchu a nulový při neúspěchu [32].

Doplňující informaci k návratové hodnotě dokáže poskytnout funkce `GetLastError`. Pro potřeby této práce je důležitá zejména hodnota `0x5` (`ERROR_ACCESS_DENIED` [33]), která je vracena v případě, že odeslání zprávy bylo zablokováno *User Interface Privilege Isolation* (UIPI) [31, 32].

## 1.5 Filtrování

První možností, která se nabízí v případě, že aplikace potřebuje některé *window messages* explicitně povolit nebo blokovat, je integrované filtrování ve funkci `GetMessage` skrze parametry `wMsgFilterMin` a `wMsgFilterMax`. Ty umožňují specifikovat nejnižší a nejvyšší číselný identifikátor zpráv, které mají být z fronty vyjímány. V případě, že jsou oba tyto parametry nastaveny na nulu, nedochází k žádnému filtrování. Jedinou výjimkou je zpráva `WM_QUIT`, která je z fronty vybrána vždy bez ohledu na nastavení parametrů `wMsgFilterMin` a `wMsgFilterMax` [24].

Vzhledem k tomu, že tato funkcionalita umožňuje stanovit rozsah zpráv vybíraných z fronty, bylo by ji teoreticky možné použít pro zabezpečení programu proti útokům typu *Shatter*. Nicméně jako mnohem praktičtější přístup k filtrování zpráv se jeví přímá interakce s UIPI filtrem skrze funkce `ChangeWindowMessageFilter` a `ChangeWindowMessageFilterEx`, které jsou rozebírány později v sekci 4.3. Funkce `GetMessage` nachází své uplatnění spíše v případech, kdy je potřeba upravit pořadí vybírání zpráv z fronty (popsáno v sekci 1.1).

## 1.6 Problémy

Protože byl celý systém předávání zpráv mezi programy běžícími v operačním systému navržen v dobách, kdy ve fázi návrhu nebyl příliš kladen důraz na bezpečnost, obsahuje architektura tohoto systému zásadní nedostatky.

V sekci 1.2 byl věnován prostor strukturám, ve kterých jsou zprávy předávány. Za povšimnutí stojí, že tyto struktury v žádném z případů neobsahují informaci o tom, kdo je původcem zprávy. Cílový program tedy nemá šanci rozlišit, zda zprávu odeslal důvěryhodný subjekt, jako např. operační systém, nebo subjekt s nekalým úmyslem, jako útočník. Systém předávání zpráv neposkytuje programu, který zprávy přijímá, infrastrukturu nutnou k tomu, aby se rozhodl, zda je zprávu bezpečné zpracovat. Tento fakt doplněný o informaci o tom, že existují zprávy jako `WM_TIMER`, které jako jeden ze svých parametrů obsahují adresu funkce, jež se v cílovém programu použije jako *callback* [34], poukazuje na to, že se systém předávání zpráv dá zneužít v celé řadě útoků.

Druhou částí problému je, že v dobách před zavedením *User Interface Privilege Isolation*, tedy před operačním systémem Windows Vista, neexistoval žádný mechanismus, který by omezoval možnost programů posílat oknům *window messages*. Jinými slovy, libovolný program mohl libovolnému oknu poslat libovolnou zprávu. [35]

Je tedy snadné si představit situaci, kdy program, který běží s nízkými oprávněními, pošle zprávu `WM_TIMER` programu, který běží s administrátorskými oprávněními. Adresa funkce v parametru `WM_TIMER` bude spuštěna s oprávněními privilegovaného programu a dochází k *privilege escalation*. Tímto způsobem vznikají útoky typu *Shatter*.

# Kapitola 2
# Útoky typu *Shatter*

Prapůvodní útok *Shatter* (CWE-422 [36]) se poprvé objevil v květnu roku 2002, kdy Chris Paget, známý také jako *Foon*, na svém webu zveřejnil článek o útoku, který zneužívá toho, že libovolná aplikace může libovolnému oknu poslat *window message* bez ohledu na to, jestli dané okno vlastní. Tento útok pojmenoval *Shatter* a to kvůli tomu, že z jeho pohledu byla v době vydání článku neopravitelná a doslova tříštila bezpečnostní prvky Windows. [37]

## 2.1 Předpoklady

Prvním předpokladem pro provedení úspěšného útoku je správná volba operačního systému. Ten je nutné zvolit tak, aby neobsahoval žádný z *patchů*, kterými Microsoft později upravil chování *window messages*, a nepoužíval technologii *User Interface Privilege Isolation*. Mezi zranitelné operační systémy patří [38]:

- Microsoft Windows NT 4.0;
- Microsoft Windows NT 4.0, Terminal Server Edition;
- Microsoft Windows 2000;
- Microsoft Windows XP (bez nainstalovaného *Service Packu*).

Druhým předpokladem je, že na *interactive desktop* běží proces nebo služba se zvýšenými oprávněními [38] (např. pod uživateli `Administrator` nebo `LocalSystem`). To, že proces nebo služba běží na *interactive desktop*, umožňuje ostatním procesům tomuto procesu nebo službě posílat *window messages*.

K porozumění *interactive desktop* je třeba nejprve definovat pojem *window station*. *Window station* se rozumí objekt, který zaobaluje globální informace, jako je obsah schránky[1] nebo tabulka atomů, a jeden nebo více *desktop* objektů [39]. *Interactive window station* je potom ta jediná *window station*, která je viditelná uživateli a obsahuje myš, klávesnici a zobrazovací zařízení [39]. Pouze jeden *desktop* objekt v rámci *interactive window station* může být aktivní. Tento objekt nazýváme *interactive desktop*. *Interactive desktop* obsahuje zobrazovaný obsah jako okna nebo menu a přijímá vstupy od uživatele [38].

Z toho vyplývá, že útočník může úspěšný útok provést pouze v případě, že má přístup k *interactive desktop*, tedy je schopný se do operačního systému přihlásit [40]. To ale neznamená, že by útok nemohl být proveden vzdáleně, např. přes *Terminal Services* dnes známé jako *Remote Desktop Services* [41].

[1] V kontextu této práce je pojem schránka využíván jako synonymum k anglickému výrazu „*clipboard*.“

## 2.2 Rekonstrukce a mechanismus

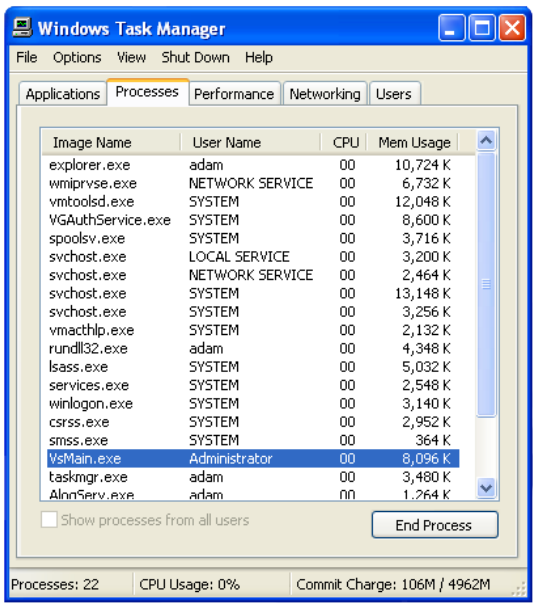
V článku zveřejnil Paget poměrně detailní postup jak útok provést včetně aplikace, která implementuje odesílání zpráv cílovému procesu a podstatně tak útok zjednodušuje. Součástí archivu, který Paget k článku přiložil, je i její zdrojový kód, jehož části jsou v této sekci rozebrány, a *payload*, který slouží pro dokončení útoku a získání *reverse shell* se zvýšenými oprávněními. [37]

Princip útoku *Shatter* v podobě, ve které jej zveřejnil v roce 2002, zneužívá problémů, které byly vytyčeny v sekci 1.6, tedy toho, že cílový program není schopný rozlišit mezi odesílateli příchozích zpráv a nezbývá mu nic jiného, než všechny zprávy zpracovávat stejným způsobem bez ohledu na to, jaký proces byl jejich původcem. Druhou částí problému je, že libovolný proces může libovolnému oknu odeslat jakoukoliv zprávu [35].

K demonstraci útoku používá Paget operační systém Windows 2000 Professional [37]. My při rekonstrukci použijeme o něco novější Windows XP ve verzi 5.1.2600, který nemá nainstalovaný žádný *Service Pack*.

Útok začíná vhodnou volbou cíle. Je-li cílem útočníka získat zvýšená oprávnění, musí identifikovat proces, který, jak bylo nastíněno v sekci 2.1 o předpokladech útoku, zobrazuje okno na *interactive desktop* a zároveň má zvýšená oprávnění. K nalezení takového procesu může posloužit vestavěná funkcionalita operačního systému Windows, *Task Manager*. Jak je vidět na obrázku 2.1, v záložce *Processes* ve sloupečku *User Name* zobrazuje pro každý proces uživatele, pod kterým je spuštěn.

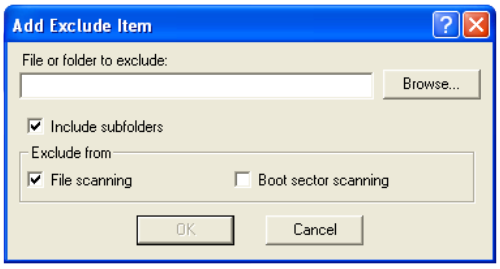
**Obrázek 2.1** Zobrazení vlastníků běžících procesů v programu *Task Manager* [42]




Paget jde pro účely demonstrace jinou cestou a sám si do operačního systému doinstalovává program, na který bude útočit. V jeho verzi jde o *Network Associates VirusScan* ve verzi 4.5.1 [37], my použijeme *McAfee VirusScan* verze 5.1 [43], který spustíme pod uživatelem `Administrator`.

Druhým krokem je do adresního prostoru zvoleného procesu umístit *payload* v podobě instrukcí, které budou po zneužití zranitelnosti vykonány. Paget tento problém řeší již v prvním kroku, a to vhodnou volbou zranitelné aplikace [37]. Jak v *Network Associates VirusScan*, tak v *McAfee VirusScan* nalezneme políčko, do kterého lze zadávat text. Cílem útoku v naší rekonstrukci bude políčko (na obrázku 2.2), které se nachází na obrazovce přístupné pod tlačítky *Scan*, *Settings*, *Exclusion* a *Add...*

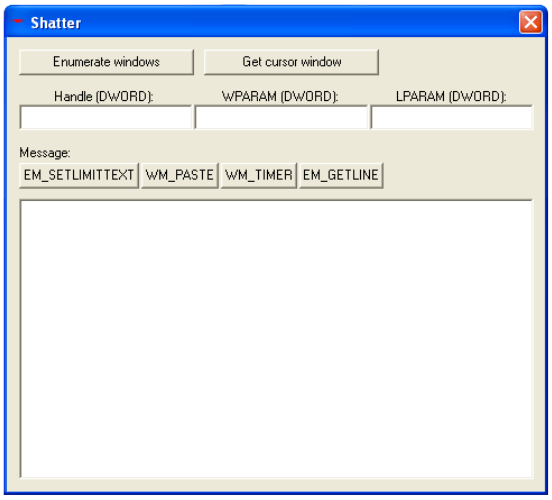
**Obrázek 2.2** Okno obsahující textové pole, na které je útočeno v rekonstrukci útoku *Shatter* [43]




V tento moment je využita aplikace `shatter.exe` [44] (na obrázku 2.3) zveřejněná jako součást archivu, který je přiložen k Pagetově článku. Povšimněme si, že aplikaci spouštíme pod lokálním uživatelem a neběží tedy se zvýšenými oprávněními. Pomocí následující posloupnosti akcí [37] získáme *handle* políčka, na které útočíme:

- stiskneme tlačítko *Get cursor window*;
- kurzorem najedeme na cílové políčko;
- stiskneme mezerník;

**Obrázek 2.3** Rozhraní aplikace `shatter.exe` [44]




Následně by se v poli ve spodní části aplikace mělo zobrazit hexadecimální číslo, které je číselným vyjádřením *handle* políčka. Toto číslo si můžeme rovnou přepsat do kolonky *Handle*. Z výpisu kódu 2.1 zobrazujícího zdrojový kód aplikace `shatter.exe`, který je součástí zmiňovaného archivu, je vidět, že této funkcionality dosahuje použitím dvou funkcí z Windows API. Nejdříve pomocí funkce `GetCursorPos` získá souřadnice kurzoru [45], které poté použije jako parametr ve funkci `WindowFromPoint`, jejíž návratová hodnota je *handle* okna, které se na těchto souřadnicích vykresluje [46].

Jakmile máme k dispozici *handle* políčka, můžeme použít rozhraní aplikace `shatter.exe` k posílání *window messages*. Nejprve podle Pagetova postupu [37] použijeme zprávu `EM_SETLIMITTEXT`,

**Výpis kódu 2.1** Implementace dohledání *handle* okna podle pozice kurzoru v aplikaci `shatter.exe` [44]

```
void CShatterDlg::OnButton5()
{
        EmptyBox();
        POINT CursorPos;
        if (GetCursorPos(&CursorPos))
        {
                HWND CursorWindow = ::WindowFromPoint(CursorPos);
                // ...
        }
}
```

která jako `wParam` bere maximální počet znaků, který může být do daného prvku zadán. V případě, že je `wParam` nastaven na nulu, je tento počet nastaven na 64 000 znaků [47]. Prostřednictvím aplikace `shatter.exe` pošleme tuto zprávu tak, že do políčka `WPARAM` zadáme hodnotu `0`, políčko `LPARAM` necháme prázdné a stiskneme tlačítko `EM_SETLIMITTEXT`. Tímto způsobem sejmeme z cílového políčka omezení na délku zadávaného textu, které by mohlo negativně ovlivnit náš *payload*.

Rozborem zdrojového kódu aplikace ve výpisu kódu 2.2 zjistíme, že Paget k odeslání zprávy volil funkci `PostMessage`, která vytvoří *queued* zprávu, a na místě parametru `Msg` má napevno vloženou hodnotu `EM_SETLIMITTEXT`.

**Výpis kódu 2.2** Implementace odeslání zprávy `EM_SETLIMITTEXT` v aplikaci `shatter.exe` [44]

```
void CShatterDlg::OnButton9()
{
        if (!::PostMessage((HWND)WindowHandle,EM_SETLIMITTEXT,m_wparam,
                m_lparam))
                MessageBox("Message failed!","Error:",
                        MB_ICONWARNING|MB_OK);
}
```

Nyní použijeme *payload*, který Paget přibalil v souboru `sploit.bin` [44]. V článku uvádí, že sám není jeho autorem a přisuzuje ho hackerovi s přezdívkou *Dark Spyrit* [37]. Obsah tohoto souboru, byť v lehce pozměněné podobě, lze dohledat jako součást *exploitu*, který zneužívá historickou zranitelnost typu *buffer overflow* v Microsoft IIS 5.0 [48]. V hlavičce souboru, který *exploit* obsahuje, je skutečně podepsán *Dark Spyrit* [48]. Paget uvádí, že *payload* je konstruován tak, aby otevřel *reverse shell* na *localhost* adrese a *portu* 123 [37].

Soubor `sploit.bin` tedy otevřeme v libovolném textovém editoru a zkopírujeme do schránky. V našem případě bychom nyní mohli kliknout pravým tlačítkem myši a zvolit *Paste* nebo použít klávesovou zkratku `CTRL+V`. Nicméně pro případ, že toto udělat nemůžeme, uvádí Paget obecnější postup s použitím zprávy `WM_PASTE`. V aplikaci `shatter.exe` nastavíme jak `WPARAM`, tak `LPARAM` na nulu a stiskneme tlačítko `WM_PASTE`. To způsobí odeslání stejnojmenné zprávy, která zapříčiní, že obsah schránky je vložen do cílového políčka [49]. Bokem si připravíme *Netcat* [50] a pomocí příkazu `nc -lp 123` jej nastavíme, aby poslouchal na adrese `localhost:123`.

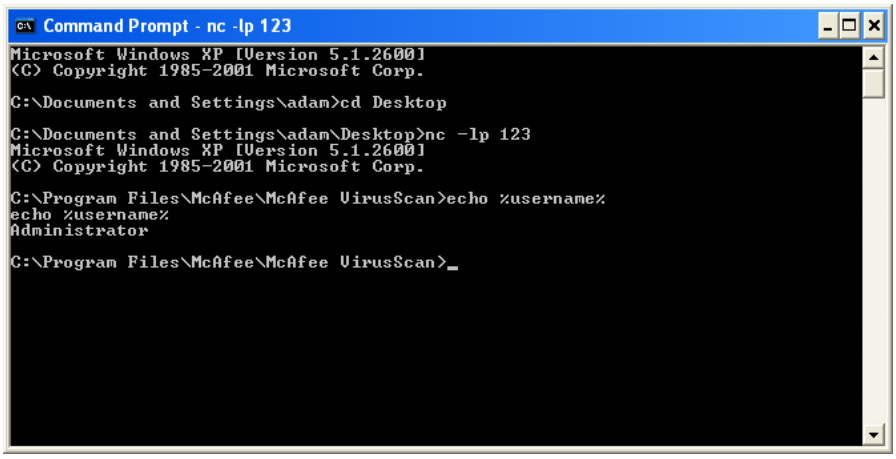
V následujícím kroku potřebujeme zjistit adresu *payloadu* v paměti cílového procesu. Paget postupuje tak, že spustí WinDbg [51], který připojí k cílovému procesu (v našem případě jde o `VsMain.exe`) a pomocí příkazu `s -a 00000001 10000000 "FOON"` vyhledá řetězec `FOON`, který

se nachází na začátku *payloadu* [37]. Protože připojení *debuggeru* je privilegovaná operace, musíme WinDbg spouštět s administrátorskými oprávněními.

Při reálném útoku bychom samozřejmě taková oprávnění neměli a museli bychom použít alternativní postup. Paget uvádí, že by postačilo získat *load* adresu cílového programu a zjistit vzdálenost *payloadu* v paměti [37]. Tímto způsobem bychom mohli útočit na všechny operační systémy se stejnou verzí cílového programu [37]. Druhou možností by bylo použít techniku `NOP` *sled*, kdy před *payload* vložíme dlouhou posloupnost instrukcí `NOP`, která nám usnadní se trefit do adresy *payloadu* [37]. Tato technika se používá i v *payloadu* od *Dark Spyrit* [48].

*Debugger* můžeme nyní odpojit a zjištěnou adresu vložíme do parametru `LPARAM`. Hodnota `WPARAM` může být libovolná [37]. Protože *payload* obsahuje `NOP` *sled* od délce 1 024 *bytů*, přičteme k adrese 512 *bytů*, abychom se trefili přímo do jeho prostředku [37]. Útok dokončíme tím, že stiskneme tlačítko `WM_TIMER`, které odešle stejnojmennou zprávu. *Netcat* by měl zachytit příchozí *reverse shell* a pomocí příkazu `echo %username%` si můžeme ověřit, že, jak lze vidět na obrázku 2.4, *shell* běží pod uživatelem `Administrator`.

**Obrázek 2.4** *Reverse shell* v programu *Netcat* získaný jako výsledek útoku *Shatter* [50]




Zpráva `WM_TIMER` je zpráva, která je procesu poslána v případě, že vyprší časovač vytvořený pomocí funkce `SetTimer`. Jako `wParam` by měla obsahovat identifikátor časovače a jako `lParam` ukazatel na funkci, která má být zavolána při jeho vypršení. Problém tedy spočívá v tom, že libovolný proces může libovolnému procesu tuto zprávu poslat a na straně příjemce zprávy se zpráva bude jevit, jako by byla poslána jakožto výsledek expirace časovače vytvořeného pomocí `SetTimer`. Příjemce pod tímto dojmem začne vykonávat funkci, kterou může útočník podvrhnout do parametru `lParam`. Tato funkce by měla mít signaturu vyobrazenou ve výpisu kódu 2.3. [34]

**Výpis kódu 2.3** Funkční signatura *callback* funkce použitelné ve zprávě `WM_TIMER` [34]

```
TIMERPROC Timerproc;

void Timerproc(HWND unnamedParam1, UINT unnamedParam2,
        UINT_PTR unnamedParam3, DWORD unnamedParam4);
```

## 2.3 Reakce Microsoftu

Před zveřejněním svého článku se Paget pokoušel prostřednictvím emailu upozornit Microsoft *Security Response Center* na nalezenou zranitelnost a vyjádřit její závažnost. Zástupce Microsoftu však v odpovědi z valné části zamítl, že by se mělo jednat o problém, přestože Microsoft ví, že takové zranitelnosti v jeho operačním systému existují. [52]

Jako hlavní argument zástupce Microsoftu používá, že Microsoft dlouhodobě programátory odrazuje od používání *interactive services* [52], což jsou služby, které zobrazují okna na *interactive desktop* [53]. Tím naráží na aplikaci *Network Associates VirusScan*, kterou Paget při demonstraci používá. Ta totiž současně běží pod uživatelem `LocalSystem` a zobrazuje interaktivní okno na ploše. Microsoft tvrdí, že veškerá okna na *interactive desktop* si jsou vzájemně rovna a mohou potenciálně získat stejnou úroveň oprávnění, jako má aplikace s nejvyššími oprávněními zobrazující okna na *interactive desktop* [40]. Zodpovědnost za bezpečné zpracování *window messages* by podle Microsoftu měla náležet programátorovi, který danou aplikaci vyvíjí [52].

Jak je nastíněno v sekci 1.6 o problémech *window messages*, je požadavek na to, aby se programátor zabýval zabezpečením své aplikace proti všem zranitelnostem plynoucím z problémů *window messages*, téměř nesplnitelný. Protože neexistuje způsob, jak konkrétní zprávu přiřadit procesu, který ji odeslal, nezbylo by programátorovi než zablokovat všechny zprávy typu `WM_TIMER`. Jak je ale demonstrováno v sekci 2.4 o dalších variantách útoku *Shatter*, `WM_TIMER` není jedinou zneužitelnou zprávou [54], a proto by takové opatření nebylo efektivní.

Ve své reakci [52] se Microsoft dále odvolává na svůj článek o desateru bezpečnosti [55], konkrétně na bod, který tvrdí, že pokud má útočník fyzický přístup ke stroji oběti, pak již nemůžeme považovat oběť za vlastníka tohoto stroje. Jinými slovy tvrdí, že úspěšné provedení útoku *Shatter* je podmíněné fyzickým přístupem. Toto tvrzení není zcela pravdivé, protože, jak již bylo zmíněno v sekci 2.1, lze útok provést i za pomocí vzdáleného přístupu s pomocí *Terminal Services*.

O několik měsíců později Microsoft své stanovisko přehodnotil a vydal bezpečnostní bulletin [38], ve kterém uznává, že se jedná o závažný problém. V textu bulletinu bere zpět své tvrzení o rovnocennosti oken na *interactive desktop* a tvrdí, že situace je o něco komplikovanější. Podle nového názoru Microsoftu mohou na *interactive desktop* koexistovat okna s různými úrovněmi oprávnění, dokud privilegovaná okna správně kontrolují a zpracovávají příchozí požadavky, jako např. *window messages*. Chyba ve `WM_TIMER` podle Microsoftu snižovala efektivitu těchto kontrol. Opětovně ale upozorňuje, že spouštění privilegovaných procesů s okny na *interactive desktop* má značná rizika. [38]

V rámci bulletinu Microsoft přiznává, že se sám neřídil svými doporučeními ohledně privilegovaných procesů na *interactive desktop* [38] a že ve Windows existovalo hned několik *interactive services*. Tuto skutečnost potvrzuje i Paget v druhém díle svého článku [41], kde některé takové služby identifikuje.

Společně s bulletinem vychází také *patch*, který mění chování `WM_TIMER` tak, aby prostřednictvím této zprávy nemohly být volány funkce, které nebyly před tím správně zaregistrovány. K tomu Microsoft uvádí, že revidoval a pozměnil zpracování *window messages* v *interactive services*, které běží se zvýšenými oprávněními. Podobná opatření zařadil Microsoft i do *Service Pack* 1 pro Windows XP. [38]

**Tabulka 2.1** Časová osa událostí kolem útoku *Shatter* [52, 37, 40, 41, 38]

| Datum | Událost |
|---|---|
| srpen 2002 | Paget posílá email na Microsoft *Security Response Center*. |
| srpen 2002 | Paget vydává první článek o *Shatteru*. |
| září 2002 | Microsoft vydává odpověď na Pagetův článek. |
| září 2002 | Paget vydává druhý článek o *Shatteru*. |
| prosinec 2002 | Microsoft vydává bezpečnostní bulletin a *patch* na *Shatter*. |

## 2.4 Varianty

Útoky typu *Shatter* nemusí být nutně provedeny pouze za pomocí zprávy `WM_TIMER`. `WM_TIMER` je pouze jednou ze skupiny zpráv, které umožňují odesílateli specifikovat adresu funkce, která je následně zavolána z cílového procesu jako *callback*. Navíc existuje další skupina *window messages*, které ukazatele na *callback* předávají jako atribut ve struktuře uvnitř jejich parametru. [56]

Zcela odlišná větev útoků typu *Shatter* je založena na zprávách, které umožňují zapisovat data do paměťového prostoru cílového procesu. Některé takové zprávy provádí zápis řízeným způsobem na konkrétní místo, které odesílatel nemůže ovlivnit, jako např. titulek v hlavičce okna. U jiných zpráv se ale ukázalo, že umožňují přímo specifikovat adresu, kam mají být data zapsána, která není nijak operačním systémem kontrolována. [56]

### 2.4.1 Zprávy obsahující *callback*

Variantě útoku *Shatter*, která zneužívá zprávu, která v nějaké podobě umožňuje specifikovat funkci následně použitou jako *callback* v cílové aplikaci, je věnována většina předchozích sekcí této kapitoly. Přesně na tomto mechanismu stojí *Shatter* v podobě, ve které ho zveřejnil Paget v roce 2002.

Poté, co byl na konci roku 2002 vydán *patch*, který opravil zranitelnost ve zprávě `WM_TIMER`, bylo nalezeno ještě několik dalších zpráv pracujících s ukazateli na funkce. Nicméně většina těchto zpráv je spjata s určitým ovládacím prvkem, jako např. *list-view* nebo *rich edit*, a jejich zneužití je tedy podmíněno tím, že cílové okno tento ovládací prvek obsahuje. Mezi zprávy (mimo `WM_TIMER`), které obsahují ukazatel na *callback* přímo v jednom ze svých parametrů, patří [56, 57]:

- `LVM_SORTITEMS`;
- `LVM_SORTITEMSEX`;
- `EM_SETWORDBREAKPROC`;
- `EM_SETWORDBREAKPROCEX`.

Zprávy `LVM_SORTITEMS` [58] a `LVM_SORTITEMSEX` [59] jsou zprávy, které jsou spjaty s ovládacím prvkem *list-view*. Obě zprávy umožňují jako `lParam` předat ukazatel na funkci, která bude volána při řazení položek *list-view* pokaždé, když bude potřeba rozhodnout o relativním pořadí dvou položek. Zprávy `LVM_SORTITEMS` a `LVM_SORTITEMSEX` se vzájemně odlišují pouze v tom, zda do této funkce budou předávány hodnoty (`LVM_SORTITEMS`) nebo pouze indexy (`LVM_SORTITEMSEX`) porovnávaných prvků. *Callback* funkce by měla mít signaturu jako funkce ve výpisu kódu 2.4.

**Výpis kódu 2.4** Funkční signatura *callback* funkce použitelné ve zprávách `LVM_SORTITEMS` a `LVM_SORTITEMSEX` [58, 59]

```
int CALLBACK CompareFunc(LPARAM lParam1, LPARAM lParam2,
        LPARAM lParamSort);
```

Dvojice zpráv `EM_SETWORDBREAKPROC` [60] a `EM_SETWORDBREAKPROCEX` [61] řídí zalamování textu v ovládacích prvcích typu *edit*. Opět v parametru `lParam` umožňují předávat ukazatel na funkci, pomocí které může programátor definovat vlastní způsob zalamování. Obě zprávy se od sebe liší signaturou těchto funkcí na výpisech kódu 2.5 a 2.6.

Další potenciálně zneužitelná skupina zpráv obsahuje ukazatel na funkci ve struktuře, která je předávána jako jeden z jejich parametrů [56]:

**Výpis kódu 2.5** Funkční signatura *callback* funkce použitelné ve zprávě `EM_SETWORDBREAKPROC` [62]

```
EDITWORDBREAKPROCA Editwordbreakproca;

int Editwordbreakproca(LPSTR lpch, int ichCurrent, int cch, int code);
```

**Výpis kódu 2.6** Funkční signatura *callback* funkce použitelné ve zprávě `EM_SETWORDBREAKPROCEX` [63]

```
EDITWORDBREAKPROCEX Editwordbreakprocex;

LONG Editwordbreakprocex(char *pchText, LONG cchText, BYTE bCharSet,
        INT action);
```

- `EM_STREAMIN`;
- `EM_STREAMOUT`;
- `EM_SETHYPHENATEINFO`;
- `TVM_SORTCHILDRENCB`.

`EM_STREAMIN` [64] resp. `EM_STREAMOUT` [65] jsou zprávy, které se používají pro načtení resp. uložení dat ovládacího prvku *rich edit*. Obě funkce přijímají jako `lParam` strukturu `EDITSTREAM`. Atribut `pfnCallback` této struktury je adresa funkce, která se při načítání resp. ukládání dat bude v cílové aplikaci volat [66]. Funkce odkazovaná `pfnCallback` by měla splňovat signaturu ve výpisu kódu 2.7.

**Výpis kódu 2.7** Funkční signatura *callback* funkce použitelné ve struktuře `EDITSTREAM` [67]

```
EDITSTREAMCALLBACK Editstreamcallback;

DWORD Editstreamcallback(DWORD_PTR dwCookie, LPBYTE pbBuff, LONG cb,
        LONG *pcb);
```

Ke stejnému ovládacímu prvku jako zprávy `EM_STREAMIN` a `EM_STREAMOUT` se váže i zpráva `EM_SETHYPHENATEINFO` [68]. V parametru `wParam` přijímá odkaz na strukturu `HYPHENATEINFO`, která obsahuje odkaz na funkci [69] řídící způsob, jakým se v *rich edit* ovládacím prvku dělí slova [70]. Tato funkce musí splňovat signaturu z výpisu kódu 2.8.

Poslední z *window messages*, které obsahují ukazatel na funkci obsažený ve struktuře, je zpráva `TVM_SORTCHILDRENCB` [71]. V tomto případě se jedná o strukturu `TVSORTCB`. Ukazatel na funkci v ní obsažený se, podobně jako u `LVM_SORTITEMS` a `LVM_SORTITEMSEX`, použije pokaždé, když je při řazení položek *tree-view* ovládacího prvku potřeba rozhodnout o vzájemném pořadí dvou prvků. Funkce, na kterou je odkazováno, by měla splňovat signaturu ve výpisu kódu 2.9.

### 2.4.2 Zprávy zapisující do paměti procesu

Útok za pomocí zpráv tohoto typu stojí na nedostatečné validaci jejich vstupních parametrů. Příkladem může být zpráva `TCM_GETITEMRECT` [73], která se používá pro získání informací o ohraničujícím obdélníku ovládacího prvku typu *tab*. Tyto informace jsou předávány formou struktury `RECT` (ve výpisu kódu 2.10), na níž odkazuje parametr `lParam`.

**Výpis kódu 2.8** Funkční signatura *callback* funkce použitelné ve struktuře `HYPHENATEINFO` [70]

```
void HyphenateProc(WCHAR *pszWord, LANGID langid, long ichExceed,
        HYPHRESULT *phyphresult);
```

**Výpis kódu 2.9** Funkční signatura *callback* funkce použitelné ve struktuře `TVSORTCB` [72]

```
int CALLBACK CompareFunc(LPARAM lParam1, LPARAM lParam2,
        LPARAM lParamSort);
```

Zneužitelnost této zprávy spočívá v tom, že adresa předaná v parametru `lParam` není kontrolována [56]. Tuto adresu tedy lze nastavit na libovolnou adresu z adresního prostoru cílového procesu a opakovaným posíláním zpráv `TCM_GETITEMRECT` do něj zapisovat struktury typu `RECT`. [56].

Jelikož je možné pomocí zprávy `TCM_SETITEMSIZE` nastavovat výšku a šířku ovládacího prvku typu *tab* [75], dokáže útočník přímo ovlivňovat obsah struktury `RECT` vracené zprávou `TCM_GETITEMRECT`. V kombinaci s technikou popisovanou výše může do adresního prostoru libovolného procesu řízeně zapisovat libovolná data [56].

Prostřednictvím kombinace těchto dvou zpráv lze do paměťového prostoru procesu infiltrovat *payload* a následně stejnou metodou přepsat adresu *handleru Structured Exception Handling* (SEH) výjimek. Do úspěšného útoku pak už zbývá jen vyvolat výjimku, načež dojde k přepnutí řízení programu na adresu SEH *handleru*, která byla nahrazena adresou infiltrovaného *payloadu* a ten se začne vykonávat. Kromě toho by útočník mohl cílit na *Process Enviroment Block* (PEB) či *Thread Environment Block* (TEB) a snažit se o přepsání jejich položek [56].

`TCM_GETITEMRECT` a `TCM_SETITEMSIZE` ale není jediný pár zpráv, který lze tímto způsobem zneužít. Mezi další dvojice zpráv, s jejichž pomocí lze zkonstruovat *exploit* na stejném principu, patří [57]:

- `BCM_SETTEXTMARGIN` a `BCM_GETTEXTMARGIN`;
- `HDM_SETORDERARRAY` a `HDM_GETORDERARRAY`;
- `HDM_SETITEM` a `HDM_GETITEM`;
- `LVM_SETCOLUMNWIDTH` a `HDM_GETITEMRECT`;
- `LVM_SETCOLUMNORDERARRAY` a `LVM_GETCOLUMNORDERARRAY`;
- `LVM_SETITEM` a `LVM_GETITEM`;
- `LVM_SETITEMPOSITION` a `LVM_GETITEMPOSITION`;
- `LVM_SETITEMTEXT` a `LVM_GETITEMTEXT`;
- `PBM_SETRANGE` a `PBM_GETRANGE`;
- `SB_SETPARTS` a `SB_GETPARTS`;
- `TCM_SETITEM` a `TCM_GETITEM`.

Tento seznam však není vyčerpávající a nelze vyloučit, že jsou zneužitelné i jiné zprávy. Kromě nich existují další zprávy, které sice nemají svůj protějšek ve dvojici, ale přesto umožňují zapisovat na libovolnou adresu [56]:

**Výpis kódu 2.10** Obsah struktury `RECT` [74]

```
typedef struct tagRECT {
        LONG left;
        LONG top;
        LONG right;
        LONG bottom;
} RECT, *PRECT, *NPRECT, *LPRECT;
```

- `LVM_CREATEDRAGIMAGE`;
- `LVM_GETITEMRECT`;
- `LVM_GETNUMBEROFWORKAREAS`;
- `LVM_GETSUBITEMRECT`;
- `LVM_GETVIEWRECT`;
- `TB_GETMAXSIZE`;
- `TVM_GETITEMRECT`.

Speciálním případem zpráv, které dokáží zapisovat do paměťového prostoru cílového procesu je dvojice zpráv `CB_DIR` a `LB_DIR`. Téměř o rok později od vydání článku o původním útoku *Shatter*, v říjnu roku 2003, byla objevena zranitelnost těchto dvou zpráv dovolující *buffer overflow* (CVE-2003-0659 [76]) [57].

Zpráva `CB_DIR` [77] resp. `LB_DIR` [78] se používá k přidání souborů či adresářů do seznamu zobrazovaného ovládacím prvkem *list box* resp. *combo box*. V parametru `wParam` obou těchto zpráv lze pomocí řetězce specifikovat cestu souborů či adresářů, které mají být přidány. Zranitelnost spočívala v tom, že funkce z `User32.dll`, která tyto zprávy zpracovává, špatně kontrolovala délku zadaných řetězců [57]. Výsledkem bylo, že pomocí vhodných `CB_DIR` nebo `LB_DIR` zpráv mohl útočník přepisovat paměťový prostor cílového procesu a zařídit spuštění jím infiltrovaného kódu [79]. V případě, že cílový proces běžel se zvýšenými oprávněními, umožňovala tato zranitelnost útočníkovi infiltrovaný kód vykonat s oprávněními tohoto procesu a dosáhnout tak *privilege escalation* [79].

Ovšem, podobně jako u předchozích variant útoku *Shatter*, které nepoužívají zprávu `WM_TIMER`, je nutné, aby cílový program používal zranitelné kontrolní prvky a, jak je zmíněno výše, eskalace oprávnění je podmíněna tím, že cílový proces již se zvýšenými oprávněními běží. Jedním ze zranitelných programů byl Microsoftem dodávaný *Utility Manager* [57, 79].

## 2.5 Dopady

Nejzávažnějším dopadem útoků typu *Shatter* je *privilege escalation* [37]. To je však podmíněno tím, že cílový proces byl se zvýšenými oprávněními spuštěn. Nicméně, jak již bylo řečeno v sekci 2.3, ve Windows takové procesy existovaly a šlo dokonce o procesy služeb vyvíjených Microsoftem. Jakmile útočník získá zvýšená oprávnění, jako např. oprávnění účtu `Administrator`, má téměř neomezené možnosti. Může exfiltrovat citlivá data, zapojit stroj do *botnetu* nebo začít útočit na další stroje v segmentu lokální sítě.

Pro případ, že cílový program neběží se zvýšenými oprávněními, stále existuje několik scénářů využití útoku *Shatter*. Útočník může skrze zprávy `WM_QUIT` a `WM_DESTROY` zavírat aplikace a jejich okna a vytvořit tak *denial of service* (DoS) útok. V případě, že aplikace spoléhá na deaktivování

ovládacích prvků jako jsou tlačítka nebo políčka, může si útočník skrze *Shatter* tyto prvky povolit a získat tak neoprávněný přístup k blokovaným nebo skrytým funkcionalitám aplikace. Další forma útoku se dotýká aplikací, které používají vstup z ovládacích prvků pro generování *Structured Query Language* (SQL) dotazů. U těchto aplikací může útočník pomocí *Shatteru* provádět útoky jako SQL *injection*. Podobným způsobem by útočník mohl docílit *file inclusion* útoku, a to v případě, že aplikace používá výstupy kontrolních prvků při práci se soubory. [57]

## 2.6 Navržená opatření

Většina zdrojů [22, 54, 56, 57], které se věnují problematice *Shatter* útoků, se shoduje, že ideálním řešením by bylo, aby dle doporučení Microsoftu nezobrazovaly aplikace běžící se zvýšenými oprávněními okna na *interactive desktop*. Nicméně vzhledem k tomu, že sám Microsoft svá doporučení porušil [38] a v obecném případě nelze zaručit, že se vývojáři budou tímto doporučením řídit, nabízí i několik dalších řešení.

Tyler Close, Alan H. Karp a Marc Stiegler, autoři článku s názvem *Shatter-proofing Windows* [54], si tvrzení Microsoftu o tom, že oprávnění každého procesu běžícího na *interactive desktop* jsou v důsledku způsobu fungování *window messages* rovna oprávněním procesu s nejvyššími oprávněními, který na *interactive desktop* běží [38], vykládají jako výzvu k tomu, aby privilegované procesy byly odděleny od těch neprivilegovaných [54].

Toho se snaží docílit nejprve tak, že vytváří separátní *desktop* objekty v rámci jedné *window station*. Zde však naráží na problém, neboť každé vlákno se pomocí funkce `SetThreadDesktop` může přesunout do libovolného *desktop* objektu, dokud k němu drží *handle* [54]. Vzhledem k tomu, že Windows na *interactive window station* používá pro *interactive desktop* vždy jméno `Default` [6], není těžké pomocí tohoto jména *handle* dohledat.

Druhou možností, kterou Close *et al.* uvádí, je oddělit procesy na úrovni *window station*. Tento přístup ale naráží na to, že pouze *interactive window station* je viditelná uživateli a vzhledem k tomu, že okna se nemohou přesouvat z jedné *window station* do druhé, neexistuje způsob, jak s privilegovanými procesy odsunutými do jiné *window station* efektivně komunikovat. [54]

Autoři článku *Shatter-proofing Windows* ještě nastiňují další dvě řešení, a to s použitím virtualizace, nejprve za pomocí *full virtualization* skrze nástroje vyvíjené firmou VMWare a poté pouze za pomocí virtualizace operačního systému skrze nástroje jako Xen nebo Virtuozzo. Obě ale následně zamítají. V prvním případě je na vině náročnost na hardware a drahé licencování a v případě druhém fakt, že většina v té době rozšířených operačních systémů nebyla uzpůsobena k provozu virtuálních operačních systémů a musela by být upravena. [54]

Řešení, ke kterému se nakonec uchylují, je využití *job objectů* [54]. *Job object* je funkcionalita operačního systému Windows, která umožňuje sloučit několik procesů do jedné skupiny a spravovat je jako celek [80]. *Job objectu* přiřazují příznak `JOB_OBJECT_UILIMIT_HANDLES`, který zamezí procesům v něm obsaženým v tom, aby používaly *handle* vlastněný libovolným procesem mimo *job object* [54]. V navrhovaném řešení tedy počítají s tím, že by nedůvěryhodné programy byly uzavřeny v omezeném *job objectu* a nemohly by tak ovlivňovat zbytek programů, který by běžel mimo něj.

Jiní autoři, konkrétně Oliver Lavery [22] a Brett Moore [57], se shodují na tom, že by procesy mohly příchozí zprávy filtrovat a odstranit zneužitelné zprávy. Zde vznikají první náznaky myšlenky, která je později zhmotněna v podobě *User Interface Privilege Isolation*. Lavery však poukazuje na fakt, že z technického hlediska bylo v té době pomocí nástrojů Windows API poměrně obtížné docílit kýženého efektu a upřednostnil by, aby *window messages* obsahovaly informaci o jejich odesílateli [22].

Filtrování *queued* zpráv, tedy zpráv, které jsou na úrovni *message loop* vybírány z fronty a až poté předávány do *window procedure*, je triviální. Problém nastává u *non-queued* zpráv a to zejména v případě, že jsou adresovány ovládacím prvkům jako např. *list box*. V takovém případě totiž nemá programátor kontrolu nad jejich zpracováním [22]. To je ale podle Laveryho

řešitelné pomocí *sub-classingu* těchto kontrolních prvků, s jehož pomocí je možné předefinovat jejich *window procedure* vlastní funkcí [22].

Na závěr svého článku Lavery uvádí, že filtrování zneužitelných zpráv rozhodně nemůže být cestou k perfektnímu zabezpečení aplikace [22], v čemž se shoduje s Moorem [57]. Ten navrhuje, aby procesy namísto *window messages* ke komunikaci používaly jiné metody, jako např. *remote procedure calls* (RPC), sokety, *named pipes* nebo *Component Object Model* (COM) [57].

Z názorů všech těchto autorů vyplývá, že ideální řešení problému by zahrnovalo rozsáhlejší zásah do architektury operačního systému Windows. Řešení, která autoři představili s pomocí v té době dostupných prostředků Windows API, byla sice funkční, ale často byla poutána náročnou implementací, sníženým komfortem uživatele nebo tím, že nefungovala ve všech případech. Řada z autorů se pozastavuje nad myšlenkou o filtrování *window messages*, kterou však zamítají obvykle z toho důvodu, že Windows API neposkytuje prostředky, které by bez větší námahy tuto funkcionalitu poskytovaly. Microsoft se k této myšlence o pár let později vrací a vytváří efektivní opatření proti útokům typu *Shatter* v podobě *User Interface Privilege Isolation*.

# Kapitola 3

# *Mandatory Integrity Control*

*Mandatory Integrity Control* je jeden z bezpečnostních prvků operačních systémů Windows, který byl přidán s verzí Windows Vista [81]. Mezi jeho hlavní cíle patří rozlišit objekty operačního systému (procesy, soubory, registrové klíče), které vznikly pod identitou stejného uživatele, na důvěryhodné a nedůvěryhodné [82]. Stejné hodnocení, nazývané *integrity level*, pak přiřazuje i subjektům operačního systému (uživatelé, skupiny) a omezuje vzájemné interakce, jak typu subjekt–subjekt, tak typu subjekt–objekt, a to zejména v případě, že se jedná o nedůvěryhodný subjekt či objekt [83]. Přestože MIC umožňuje kategorizovat procesy dle jejich důvěryhodnosti, nedokáže poskytnout plnou izolaci a nemělo by k němu být přistupováno jako k aplikačnímu *sandboxu* [83].

Úroveň ochrany, kterou *Mandatory Integrity Control* zavádí, přišla v reakci na situace, kdy se procesy s různými úrovněmi oprávnění snažily vzájemně ovlivnit s cílem dosáhnout *privilege escalation* [83]. Do této kategorie spadají mimo jiné i útoky typu *Shatter*. Navíc se MIC soustředí ještě na ochranu objektů souborového systému a brání neautorizované modifikaci či smazání dat [83].

*Mandatory Integrity Control* čerpá z velké části z Biba modelu, což je bezpečnostní model navržený Kennethem J. Bibou v roce 1975 [84]. Již v té době přišel Biba s nápadem rozřazovat subjekty a objekty dle jejich důvěryhodnosti nebo, v jeho podání, integrity. Cílem Biba modelu bylo předejít tomu, aby subjekty ovlivňovaly data s vyšší úrovní integrity, a situacím, kdy docházelo k *information disclosure*, tedy zpřístupnění citlivých informací neoprávněným subjektům [84]. MIC sice stojí na stejných základech jako Bibův model, nicméně existuje mezi nimi i celá řada odlišností, zejména pak v interakcích mezi subjekty napříč různými úrovněmi integrity [83]. Microsoft rozšiřuje principy, které Biba vybudoval, a od řízení toku informací a prevence *information disclosure* se odchyluje k obecnějšímu způsobu omezení vzájemných interakcí mezi subjekty a objekty.

## 3.1 Přidružené pojmy

V rámci bezpečnostního subsystému operačního systému Windows se používá celá řada termínů. Objekty resp. subjekty, které byly zmíněny v úvodu této kapitoly, nazývá Microsoft *securable objects* (SO) resp. *trustees*.

*Securable objectem* se rozumí jakýkoliv objekt, kterému může být přiřazen *security descriptor*. Ve Windows může být *securable objectem* každý pojmenovaný objekt a některé nepojmenované objekty. O některých takových objektech bylo již v tomto textu pojednáno. Patří mezi ně např. soubory, adresáře, *named pipes*, *job objects*, ale hlavně procesy a vlákna.

Zmiňovaný *security descriptor* uchovává informace o příslušném *securable objectu*. Ve Windows API je reprezentován pomocí struktury `SECURITY_DESCRIPTOR` [85]. Mezi tyto informace patří [82]:

- číslo revize – verze bezpečnostního modelu používaného *Security Reference Monitorem* (SRM) v době vytvoření *security descriptoru*;
- příznaky – volitelné modifikátory měnící charakteristiky *security descriptoru*;
- *security identifier* (SID) vlastníka;
- *security identifier* primární skupiny;
- *discretionary access control list* (DACL) – identifikuje *trustees* a typy jejich přístupů, které jsou buď povoleny a nebo odepřeny;
- *system access control list* (SACL) – identifikuje *trustees* a typy jejich přístupů, které jsou zapisovány do systémového bezpečnostního logu.

Za *trustee* se ve Windows považuje účet, skupina nebo *logon session*, na které se vztahují *access control listy* (ACL). Každá *access control entry* (ACE), ze kterých se *access control list* skládá, používá *security identifier* k identifikaci konkrétního *trustee* [86]. *Security identifier trustee* se společně s dalšími informacemi, jako jsou SID skupin, *logon* SID nebo výchozí DACL používaný pro soubory, které *trustee* vytváří, ukládá uvnitř *access token*u [87]. Procesy, které jsou jménem *trustee* spouštěny pak obsahují jeho kopii [88].

*Security Reference Monitor* je komponenta operačního systému Windows, která tvoří infrastrukturu nutnou pro provádění kontrol přístupových oprávnění. Jeho součástí je algoritmus `SeAccessCheck` rozebíraný v sekci 3.3, který rozhoduje o povolení či odepření přístupu postupným procházením záznamů *access control listů* a dalších parametrů ze *security descriptoru*. Jde o jedinou komponentu bezpečnostního subsystému, která má vysoká oprávnění a běží v *kernel mode*. [89]

## 3.2 *Integrity level*

*Integrity level* reflektuje důvěryhodnost *securable objectu* či *trustee* [83]. Tuto informaci vnáší do procesu vyhodnocení přístupu, což částečně připomíná některé z myšlenek formulovaných při hledání řešení na útoky typu *Shatter* diskutovaných v sekci 2.6. Základním mechanismem, o který se vyhodnocování přístupů opírá, je, že procesy s nižším *integrity levelem* nemohou ovlivňovat procesy, které jsou v hierarchii integrity výše [83]. Podobně platí, že pokud se uživatel pokusí modifikovat či smazat objekt, jako např. soubor, bude zkontrolován jeho *integrity level* a pokud nebude větší nebo roven *integrity levelu* daného objektu, bude operace zamítnuta.

Důležité je poznamenat, že opačným směrem toho omezení neplatí. MIC nikterak neomezuje subjekty s vyšší integritou v tom, aby četly, modifikovaly nebo spouštěly objekty s nižší integritou. V tomto bodě se zásadně liší od Biba modelu, který se drží tzv. *no-read-down* přístupu. Je pravda, že existuje celá řada útoků, která staví na tom, že privilegovaná aplikace čte nedůvěryhodný vstup, nicméně Microsoft zde zaujímá stanovisko, že každá aplikace je zodpovědná za validaci vlastního vstupu. [83]

### 3.2.1 Implementace

K implementaci *integrity levelů* se používají výše zmiňované *security identifiery* [90, 82, 91]. Tento přístup byl zvolen za účelem zjednodušení integrace IL do stávajících datových struktur bezpečnostního subsystému Windows. Každý ze SID reprezentujících *integrity level* je ve tvaru `S-1-16-XXXX` [91], kde:

- znak `S` indikuje, že se jedná o SID;
- číslo `1` určuje verzi struktury SID (někdy nazývána revize);
- číslo `16` identifikuje *Mandatory Label Autority*, autoritu, která tento typ SID vydává;
- část `XXXX` je *relative identifier* (RID), který by měl obsahovat číslo reprezentující konkrétní *integrity level* [91].

V obecném případě jsou veškeré hodnoty, které následují po vydávající autoritě, nazývány „subautority“ [92]. Pomocí „subautorit“ je pak vyjádřena většina informací, které SID nese, jako je např. identifikátor domény či účtu [92]. Poslední ze „subautorit“ je zmiňovaný *relative identifier* [92], který v případě SID vydaného *Mandatory Label Autority* udává *integrity level*, ale má i další využití.

*Securable objectům* je SID indikující jejich *integrity level* přidělen pomocí speciální *mandatory label access control entry* (`SYSTEM_MANDATORY_LABEL_ACE` [90]) uvnitř jejich *system access control listu* [81, 83, 91]. To by mohlo být považováno za nestandardní, neboť, jak bylo zmíněno v sekci 3.1, SACL se primárně používá pro definici přístupů, které jsou auditovány či zapisovány do systémového bezpečnostního logu [82].

Zmiňovaná struktura `SYSTEM_MANDATORY_LABEL_ACE` (ve výpisu kódu 3.1) obsahuje strukturu `ACE_HEADER`, masku a SID [90]. `ACE_HEADER` se používá k identifikaci typu konkrétní *access control entry*, v tomto případě `SYSTEM_MANDATORY_LABEL_ACE_TYPE`, specifikaci speciálních příznaků, které mimo jiné řídí dědičnost, a k tomu nese informaci o délce konkrétní ACE [93]. Maska určuje *mandatory policy*, které se věnuje následující sekce 3.2.1.1 a poslední položka, `SidStart`, ukazuje na začátek *security identifieru* určujícího *integrity level* [90].

**Výpis kódu 3.1** Obsah struktury `SYSTEM_MANDATORY_LABEL_ACE` [90]

```
typedef struct _SYSTEM_MANDATORY_LABEL_ACE {
        ACE_HEADER  Header;
        ACCESS_MASK Mask;
        DWORD       SidStart;
} SYSTEM_MANDATORY_LABEL_ACE, *PSYSTEM_MANDATORY_LABEL_ACE;
```

Pokud jde o *trustees*, je jim *integrity level* přiřazen vždy při inicializaci jejich *access tokenu*, a to opět v podobě *security identifieru* [94] uloženého ve struktuře `TOKEN_MANDATORY_LABEL` [95] (ve výpisu kódu 3.2). V tomto případě je SID zabalen ještě ve struktuře `SID_AND_ATTRIBUTES` [96]. Ke zmiňované inicializaci *access tokenu* dochází v momentu přihlášení uživatele [94].

**Výpis kódu 3.2** Obsah struktury `TOKEN_MANDATORY_LABEL` [95]

```
typedef struct _TOKEN_MANDATORY_LABEL {
        SID_AND_ATTRIBUTES Label;
} TOKEN_MANDATORY_LABEL, *PTOKEN_MANDATORY_LABEL;
```

#### 3.2.1.1 *Policies*

*Policies* jsou používány *Mandatory Integrity Control* za účelem omezení úrovně přístupu subjektu s nižšími *integrity levely* [82, 91, 97]. Mohou být definovány jak pro *securable object*, tedy ve zmiňované struktuře `SYSTEM_MANDATORY_LABEL_ACE` pomocí atributu `Mask` [90], tak pro *trustee* v jeho *access tokenu* pomocí struktury `TOKEN_MANDATORY_POLICY` [97]. Pomocí ní lze specifikovat dva příznaky, přičemž oba jsou nastavovány automaticky pro všechny *access tokeny* [91]:

- `TOKEN_MANDATORY_NO_WRITE_UP` – znemožňuje subjektu přistupovat k jiným subjektům s vyšším IL za účelem zápisu;
- `TOKEN_MANDATORY_NEW_PROCESS_MIN` – řídí způsob, kterým je IL přiřazován procesům vytvářeným daným procesem.

Pokud je příznak `TOKEN_MANDATORY_NEW_PROCESS_MIN` nastaven v *access tokenu securable objectu* v podobě procesu, znamená to, že procesům, které tento proces vytváří, bude přiřazen nižší IL z IL vytvářejícího procesu a IL nastaveného na souboru, z nějž je vytvářený proces spouštěn. [91]

V `SYSTEM_MANDATORY_LABEL_ACE` *securable objectu* lze nastavit libovolnou kombinaci tří následujících *policies* [82, 91]:

- `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP` – znemožní *trustees* s nižším IL daný *securable object* zapisovat;
- `SYSTEM_MANDATORY_LABEL_NO_READ_UP` – znemožní *trustees* s nižším IL daný *securable object* číst;
- `SYSTEM_MANDATORY_LABEL_NO_EXECUTE_UP` – znemožní *trustees* s nižším IL daný *securable object* spouštět.

`SYSTEM_MANDATORY_LABEL_NO_WRITE_UP` je výchozím příznakem pro každý *securable object*. Pokud se však jedná o proces nebo vlákno, je do jejich *security descriptoru* přiřazen ještě navíc příznak `SYSTEM_MANDATORY_LABEL_NO_READ_UP`, aby bylo zabráněno čtení z jejich adresních prostorů [82]. O něco méně typický je `SYSTEM_MANDATORY_LABEL_NO_EXECUTE_UP`, který se využívá u COM tříd, kde blokuje přístupy za účelem jejich aktivace nebo spuštění. [82, 91]

### 3.2.2 Úrovně

Windows definuje osm *integrity levelů* [82, 91, 92] zobrazených v tabulce 3.1. Typicky se ale uvádí pouze čtyři z nich, a to *low*, *medium*, *high* a *system* [91]. Program `icacls` [98], který se používá pro jejich nastavování na souborech, navíc podporuje ještě užší množinu, konkrétně *low*, *medium* a *high*.

**Tabulka 3.1** *Integrity levely* definované Windows [82, 91, 92]

| ***Security identifier*** | ***Relative identifier*** | **Pojmenování** |
|---|---|---|
| `S-1-16-0` | `0x0` | *untrusted* |
| `S-1-16-4096` | `0x1000` | *low* |
| `S-1-16-8192` | `0x2000` | *medium* |
| `S-1-16-8448` | `0x2100` | *medium plus* |
| `S-1-16-12288` | `0x3000` | *high* |
| `S-1-16-16384` | `0x4000` | *system* |
| `S-1-16-20480` | `0x5000` | *protected process* |
| `S-1-16-28672` | `0x7000` | *secure process* |

Mimo jména *integrity levelů* uváděných v tabulce 3.1 se lze setkat ještě s *integrity levelem* označovaným jako *AppContainer*. Jak jeho jméno napovídá, je přiřazován aplikacím, které běží uvnitř *AppContainter* prostředí. Toto prostředí se používá zejména pro starší aplikace, které jsou za pomocí virtualizační technologie izolovány od zbytku procesů a mají přístup pouze k prostředkům, které jim jsou explicitně přiděleny [99]. Všechny aplikace běžící uvnitř *AppContainer* prostředí mají sice implicitně přidělen *low integrity level* [99], ale Windows navíc zajišťuje, aby ostatní aplikace s *low* IL mimo dané *AppContainer* prostředí neměly přístup k aplikaci, která

v *AppContaineru* běží [97]. K odlišení aplikací uvnitř *AppContainer* prostředí od standardních aplikací s *low* IL se používá speciální *security identifier* ve tvaru `S-1-15-2-XXXX` [100].

#### 3.2.2.1 *Untrusted*

*Untrusted integrity level* se používá pro veškeré procesy spuštěné uživatelskou skupinou *Anonymous* [82]. Nicméně vytvořit nový proces s *untrusted* IL může být podle některých zdrojů poměrně náročný úkol [101], což potvrzuje i experiment A.3.

Ukazuje se, že program s IL *untrusted*, který se pokusí načíst *knihovnu*, nebude úspěšně spuštěn. Naproti tomu *untrusted* program, který knihovnu nenačítá, funguje bez problémů. To proto, že program s IL *untrusted* nemá dostatečně vysoký IL na to, aby mohl přistoupit ke sdíleným *dynamic-link library* (DLL) souborům, které bývají opatřeny IL vyšším, než je *untrusted*. Ke spouštění procesů s těmito IL se většinou přistupuje tak, že daný proces je nejprve spuštěn s IL *medium* či *low* a až poté je jeho *integrity level* snížen na požadovanou úroveň [101].

#### 3.2.2.2 *Low*

*Integrity level low* se typicky využívá pro Internet Explorer (dnes *Edge*) *Protected Mode* [82, 91]. Programy běžící s tímto IL mají značně omezený přístup k souborům a registrům [82] a existují pro ně specializovaná úložiště jako `AppData\LocalLow` [82, 102] a klíče v registrech, jako je `KEY_CURRENT_USER\Software\LowRegistry` [81]. Podobně využíval *low* IL i *Protected View* programů z balíčku Microsoft Office. Nicméně v současných verzích Windows byl upraven a nyní používá dříve zmíněný *AppContainer* [103].

Vzhledem k tomu, jak funguje přiřazování *integrity levelů*, čemuž se věnuje sekce 3.2.3, stačí pro spuštění aplikace s *low* IL přiřadit souboru, ze kterého je spouštěna tento IL [102]. Toho můžeme docílit např. pomocí programu `icacls` [98]. Jak bylo zmíněno v sekci 3.2.2.1 o *untrusted* IL, může se stát, že aplikace nebude schopna s tímto IL běžet, neboť, nesmí zapisovat do většiny umístění v souborovém systému až na několik výjimek [102]. V obecném případě je toto chování žádoucí, protože data jsou tímto způsobem chráněna před nedůvěryhodnými aplikacemi s nižším IL, ale v případě, že je nutné nějakou aplikaci, jako je např. *Protected Mode* Internet Explorer, cíleně provozovat s nižším IL, bude pravděpodobně nutné tuto aplikaci upravit [102].

Je-li potřeba, aby aplikace běžela s *low* IL, je typicky volen přístup, kdy se aplikace rozdělí na dvě části. První část běží s *low* IL a zpracovává nedůvěryhodná data a druhá část aplikace běží se standardním IL *medium* a funguje jako prostředník vykonávající operace, ke kterým část s nízkou integritou nemá přístup. Obě části vzájemně komunikují pomocí těch prostředků *Inter-Process Communication* (IPC), které nejsou blokovány *Mandatory Integrity Control* [102]. Mezi takové prostředky patří [102]:

- *schránka*;
- *remote procedure call*;
- sokety;
- *window messages*, které proces s vyšší integritou explicitně povolil voláním jedné z funkcí pro modifikaci filtru diskutovaných v sekci 4.3;
- sdílená paměť, u které proces s vyšší integritou explicitně snížil její *integrity level*;
- COM rozhraní, které proces s vyšší integritou nastavil tak, aby mohlo být použito procesem s nízkou integritou;
- *named pipes*, u kterých proces s vyšší integritou explicitně snížil jejich *integrity level*.

Část aplikace, která běží s vyšší integritou, musí dbát na důkladnou kontrolu vstupů z části s nízkou integritou. Řada forem IPC je ale *integrity levely* ovlivněna, a to následujícími způsoby [102]:

- většina *window messages* je blokována *User Interface Privilege Isolation*;
- volání funkce `CreateRemoteThread` je blokováno;
- otevření sdílené paměti za účelem zápisu je blokováno;
- využití pojmenovaných objektů vytvořených procesem s vyšší integritou za účelem synchronizace je blokováno;
- využití běžící COM služby je blokováno.

#### 3.2.2.3 *Medium*

*Medium integrity level* je výchozí *integrity level* přiřazovaný standardním uživatelům a aplikacím [82, 94]. Takové aplikace pak mohou např. zapisovat do složky `Documents` aktuálního uživatele nebo registrového stromu `HKEY_CURRENT_USER` [81]. Z důvodu zpětné kompatibility, kterou se operační systém Windows snaží udržovat, je každému *securable objectu*, jenž v *security descriptoru* chybí záznam určující jeho IL, automaticky přiřazen právě IL *medium* [82]. Tím je vyřešena situace, kdy by nedůvěryhodné procesy, tedy procesy typicky běžící s IL *low* a nižším, mohly modifikovat *securable objecty*, kterým by chybělo přiřazení *integrity levelu* [94, 97].

#### 3.2.2.4 *Medium plus*

Poněkud záhadný je *integrity level* s názvem *medium plus*. Oficiální dokumentace o významu tohoto IL mlčí a velmi často jej ani neuvádí ve výčtech používaných IL. Přesto pro něj však existuje definovaná hodnota SID [92]. Někteří přispěvatelé na online diskuzních fórech se přiklání názoru, že *relative identifiery* určující IL nejsou definovány jako diskrétní hodnoty, ale jako intervaly [104]. To by například znamenalo, že by veškeré RID v rozsahu `0x0` až `0x0fff`, tedy všechny SID v rozsahu `S-1-16-0` až `S-1-16-4095`, byly považovány za *untrusted* či nějakou formu „*untrusted plus*“. To by částečně mohla potvrzovat oficiální dokumentace, která tvrdí, že RID jsou od sebe odděleny po `0x1000` z toho důvodu, aby umožnily používání IL o něco vyšších než je *medium* a nechaly prostor pro definici nových IL v budoucnosti [91]. Zde ale dokumentace hovoří velmi specificky pouze o IL *medium*.

Za zmínku také stojí procesy běžící se speciálním atributem *UIAccess*, kterým je přiřazován SID, k jehož RID je přičtena hodnota `0x10`. To znamená, že proces, který standardně běží s IL *medium* a má tedy RID `0x2000` resp. SID `S-1-16-8192`, bude s atributem *UIAccess* mít RID `0x2010` resp. SID `S-1-16-8208`. [91] Tyto procesy jsou detailněji rozebírány v sekci 4.2.

#### 3.2.2.5 *High*

Pro procesy běžící se zvýšenými oprávněními, jako např. procesy, které byly spuštěny skrze *User Account Control* (UAC) dialog, se standardně používá *integrity level high* [82]. Tyto procesy pak mohou přistupovat k privilegovaným složkám, jako je `ProgramFiles` nebo zapisovat do citlivých stromů v registrech jako `HKEY_LOCAL_MACHINE` [81].

#### 3.2.2.6 *System*

O úroveň výše než je *high integrity level* je postaven IL *system*. Ten se používá zejména pro služby a další systémové aplikace, jako např. `WinInit` nebo `WinLogon` [82, 94]. S tímto *integrity levelem* jsou spouštěny procesy všech *trustee*, jejichž *access tokeny* obsahují *security identifiery* `LocalSystem`, někdy označován pouze `System` (`S-1-5-18`), `LocalService` (někdy také `NT Authority`) (`S-1-5-19`) a `NetworkService` (`S-1-5-20`).

#### 3.2.2.7 *Protected process* a *secure process*

Pro *protected process* a *secure process integrity levely* je dokumentace opět poměrně řídká až skoro žádná. Není proto divu, že tyto IL jsou jen málo často uváděny v seznamech používaných IL a může se jevit, že ve Windows nejsou téměř používány [105].

*Protected process* IL by bylo možné propojit se stejnojmennou funkcionalitou operačního systému Windows. Shodou náhod byl totiž ve Windows Vista, tedy ve stejné verzi jako bylo přidáno *Mandatory Integrity Control*, zaveden nový typ procesu zvaný *protected process* [106]. Tento typ procesu se používá zejména v konotaci s *Digital Rights Managementem* (DRM), protože poskytuje zvýšenou ochranu proti neoprávněnému ovlivňování či čtení, a to i vůči procesu, který daný *protected process* vytvořil [106, 107]. Windows na takové procesy klade vyšší požadavky v podobě speciálních digitálních podpisů [82, 106]. Microsoft tuto funkcionalitu dále vyvíjí a v současných verzích Windows nachází své uplatnění při ochraně procesů *anti-malware* řešení [107]. Zdali však *protected process* skutečně využívá stejnojmenný *integrity level* je nejasné.

K *secure process* IL není dohledatelná dokumentace žádná a některé zdroje tvrdí, že se ve Windows nepoužívá a vyřazují ho ze seznamu zavedených IL [105].

### 3.2.3 Přiřazení

Žádné *securable objecty* vytvářené procesy s *integrity levelem medium* a vyšším neobsahují explicitní `SYSTEM_MANDATORY_LABEL_ACE`. Opírají se tedy o pravidlo, že *securable objecty* bez *integrity levelu* interpretuje Windows automaticky jako by měly IL *medium* a výchozí *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP`. Tímto způsobem jsou veškeré soubory nebo klíče v registrech a obecně *securable objecty* s volnými DACL chráněny před nedůvěryhodnými procesy s nízkým IL. [91]

Neznamená to však, že by *securable objecty* vytvářené uživatelem s identitou `Administrator` a tedy IL *high*, kterým je dle tohoto pravidla přiřazen IL *medium*, mohly být volně modifikovány *trustees* s nižšími IL. V tom jim zabrání výchozí nastavení DACL, které je na takové objekty automaticky aplikováno. [91]

To, že Windows nepřiřazuje *securable objectům* vytvářeným procesy s IL *medium* a vyšším explicitní IL, vychází z požadavků vznikajících kvůli UAC. V případě, že se administrátor přihlásí do Windows s vypnutým UAC, obdrží ve svém *access token*u IL *high*. Pokud by Windows přiřazoval *securable objectům* IL úměrný IL z *access tokenu* a došlo by ke spuštění UAC, ztratil by tento uživatel přístup ke svým datům. To z toho důvodu, že pokud UAC běží, je administrátorům přiřazován snížený IL *medium* namísto *high*. Jejich *access token* se v takovém případě někdy označuje jako *filtered access token* [91]

Protože procesy běžně vytvářejí dočasné soubory, musí být tato možnost zachována i pro procesy s nízkou integritou. Aby bylo pamatováno na situaci, kdy proces s nízkým IL vytvoří soubor, zavře jej a následně jej potřebuje znovu číst, je každému *securable objectu* vytvářenému procesem s IL nižším než *medium* přiřazen explicitní IL. Pokud by tomu tak nebylo a SO by obdržel výchozí IL *medium*, nebyl by proces schopen vytvářené SO opětovně číst. [91]

Když je proces vytvářen, je jeho *integrity level* určen jako minimum z IL uživatele, který proces spouští, a IL nastaveného na souboru, ze kterého je proces spouštěn [97, 82]. To znamená, že i v případě, že administrátor spustí nedůvěryhodný program, který ale má správně přiřazen IL *low*, nezíská tento program kontrolu nad operačním systémem ani přístup k jeho datům [97].

Nicméně v případě, že je proces spouštěn jiným procesem, který již běží s vyšším IL, zdědí spouštěný proces stejný IL jako měl jeho rodič. Je-li žádoucí, aby spouštěný proces běžel s nižším IL, než má jeho rodič, použije se postup, kdy nejprve duplikuje svůj *access token*, který obdržel od *trustee* při svém spuštění, následně v duplikovaném *tokenu* upraví IL a přiřadí jej nově vznikajícímu procesu. [82]

### 3.2.4 Dědění

V koncepci aplikace, která sestává ze dvou částí, jedné se standardním IL a druhé s nižším IL, např. *low*, která byla navržena v sekci 3.2.2.2, je nutné vyřešit situaci, kdy část se standardním IL vytváří soubory či jiné *securable objecty*, které mají být používány částí s nižší integritou. Bez jakékoliv úpravy totiž takové *securable objecty* budou vytvořeny bez IL a získají tedy výchozí IL *medium* [91], což znamená, že část s nižším IL nebude schopna k těmto objektům přistupovat.

Adresáře s IL *low* běžně umožňují vytváření souborů či podadresářů s různými IL [91]. V situaci diskutované výše by bylo výhodné, aby veškeré objekty, které v takové složce vznikají, automaticky získaly IL *low*. Toho lze docílit využitím atributu `AceFlags` struktury `ACE_HEADER` [91] uvnitř struktury `SYSTEM_MANDATORY_LABEL_ACE`, které se částečně věnuje sekce 3.2.1, a následujících dvou příznaků [91, 93, 82]:

- `CONTAINER_INHERIT_ACE` – podobjekty, které jsou kontejnery (např. složky) získají příslušnou ACE jako efektivní ACE (diskutováno níže);
- `OBJECT_INHERIT_ACE` – podobjekty, které nejsou kontejnery (např. běžné soubory), získají příslušnou ACE jako efektivní ACE.

Běžně lze v `AceFlags` dále definovat příznak `INHERIT_ONLY_ACE` [93, 82]. Pomocí něj můžeme ACE rozdělit na *effective* a *inherit-only* ACE. *Inherit-only* ACE, tedy ACE s příznakem `INHERIT_ONLY_ACE`, se potom neaplikuje na objekt, na němž je definovaná, ale slouží pouze k dědění vytvářenými podobjekty [93, 82]. To ale u ACE určujících IL neplatí, neboť daná integrita by vždy měla být aplikována jak na podobjekty kontejneru, tak na kontejner samotný [91].

V ACE hlavičkách se lze dále setkat s příznakem `INHERITED_ACE`, který, jak již název napovídá, pouze informuje o tom, že daná ACE byla zděděna z rodičovského objektu [93].

Pomocí příznaků řídících dědění *integrity levelů* lze zajistit konzistenci IL napříč určitou částí souborového systému a vyřešit tak problém vytyčený v úvodu této sekce. Stejné řešení používá operační systém Windows pro umístění určená pro procesy s *low* IL, jako je např. složka `AppData\LocalLow` [91]. Nicméně pro případ, kdy je nutné vytvořit objekt s jiným IL než tím, který by byl zděděn, je možné při vytváření tohoto objektu explicitně určit IL a dědění přeskočit [91]. Zde samozřejmě opět platí, že IL vytvářeného objektu nemůže být vyšší než IL subjektu, který tento objekt vytváří.

### 3.2.5 Nastavení

V sekcích 3.2.3 a 3.2.4 o přiřazování a dědění jsou popsány mechanismy, kterými jsou *integrity levely* řízeny operačním systémem. Pro případ, že uživatel chce sám nastavovat *integrity levely securable objectům*, ať se už jedná o soubory, adresáře nebo procesy, existuje několik programů s odlišnými funkcionalitami. V některých oblastech ale programy tohoto typu zcela chybí.

#### 3.2.5.1 Soubory a adresáře

Prvním programem, který lze použít pro administraci *integrity levelů* na souborech a adresářích je již dříve zmiňovaný program `icacls` [98]. Je vyvíjen přímo společností Microsoft a ve Windows je dostupný bez potřeby jakékoliv instalace. Jeho nevýhodou je, že je zaměřen zejména na DACL a co se *integrity levelů* týče, podporuje pouze úrovně *low*, *medium* a *high*. Navíc nedokáže upravovat ani číst *policies*. [98]

Alternativu poskytuje program Change mandatory label (`chml`) [108] napsaný Markem Minasim. Ten umožňuje pracovat s mnohem širší škálou IL, *untrusted*, *low*, *medium*, *high*, *system*, a navíc podporuje i manipulaci s *policies*. Další jeho výhodou je, že pomocí speciálního přepínače lze přímo upravovat *security descriptor securable objectu* prostřednictvím *Security Descriptor Definition Language* (SDDL). Pomocí SDDL je možné do *security descriptoru* zapsat

SID vyjadřující nestandardní *integrity level*, jako je např. `S-1-16-10000`, jenž se nachází někde mezi úrovní *medium* a *high*. Takový *security descriptor* by pak šel pomocí SDDL vyjádřit jako `S:AI(ML;;NW;;;S-1-16-10000)`. Za povšimnutí stojí dvojice znaků `NW`, která vyjadřuje výchozí *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP`. [108]

Za účelem analýzy toho, jak lze *integrity level* souborů a adresářů měnit, vznikla v rámci této práce aplikace `il_file`. Pomocí ní lze zobrazit, smazat a nastavit IL na konkrétní hodnotu specifikací *relative identifieru*. Využívá k tomu funkce Windows API určené pro manipulaci se *security descriptory*, *access control listy* a *access control entries*, které Microsoft řadí mezi *low-level access control* funkce [109]. Postup přečtení IL ze *security descriptoru* souboru či adresáře je následující:

- Prostřednictvím funkce `GetNamedSecurityInfo` je na základě cesty k souboru či adresáři získán *handle* k *security descriptoru* a SACL.
- Funkcí `GetAclInformation` jsou načteny informace o SACL, konkrétně o počtu ACE v něm obsažených.
- Celý SACL je procházen, dokud není nalezena `SYSTEM_MANDATORY_LABEL_ACE`.
- Pomocí funkce `ConvertSidToStringSid` je SID z `SYSTEM_MANDATORY_LABEL_ACE` konvertován do čitelné podoby.

Při zapisování nového *integrity levelu* je nejprve, stejným způsobem jako při jeho čtení, získán *handle* k *security descriptoru* a SACL a načteny jeho informace. Zbytek operace pak probíhá následovně:

- Vstupní řetězec je převeden na SID skrze funkci `ConvertStringSidToSid`.
- Je spočtena nová velikost SACL a je inicializován nový SACL pomocí `InitializeAcl`.
- Všechny ACE kromě staré `SYSTEM_MANDATORY_LABEL_ACE`, je-li v SACL již obsažena, jsou překopírovány do nového SACL pomocí funkce `AddAce`.
- Prostřednictvím funkce `AddMandatoryAce` je přidána nová `SYSTEM_MANDATORY_LABEL_ACE`.
- *Security descriptor* je zapsán do *securable objectu* pomocí funkce `SetNamedSecurityInfo`.

Poslední z operací, odebrání IL, je provedena posloupností kroků, které stejně jako u ostatních začíná voláním funkce `GetNamedSecurityInfo` za účelem získání *handle* k *security descriptoru* a SACL. Poté jsou stejným způsobem jako dříve načteny informace o SACL, který je jako při čtení procházen s cílem najít `SYSTEM_MANDATORY_LABEL_ACE`. Zbytek algoritmu je následující:

- S indexem `SYSTEM_MANDATORY_LABEL_ACE` je poté volána funkce `DeleteAce`, která ACE ze SACL odstraní.
- Jako výše je *security descriptor* zapsán do SO prostřednictvím funkce `SetNamedSecurityInfo`.

Důležité je poznamenat, že pro každou z těchto operací je nutné, aby aplikace, která ji provádí, ve svém *access tokenu* měla přiřazené oprávnění `SeSecurityPrivilege` [110, 82] známé také jako konstanta `SE_SECURITY_NAME` [111], které držiteli umožňuje číst a zapisovat SACL v *security descriptorech* [110] a provádět další operace spojené s auditováním [111]. Samozřejmostí je, že pokud se aplikace snaží *integrity level* nastavit, musí sama mít IL, který je vyšší nebo rovný tomu, který nastavuje.

Za pomocí `chml` [108] a `il_file` lze nastavovat *integrity level* na jakoukoliv hodnotu v rozmezí `0x0` (*untrusted*) až `0x3000` (*high*). V úseku `0x2000` (*medium*) až `0x3000` (*high*) je nutné využít administrátorských oprávnění, což by neměl být problém, protože stačí použít funkci *Run as*

*administrator*, která je vestavěná do Windows. Nicméně pro nastavení IL v rozmezí `0x3001` až `0x4000` (*system*) je potřeba *access token* s IL úrovně *system*, kterým disponuje např. účet `NT AUTHORITY\SYSTEM`. Této úrovně oprávnění lze dosáhnout pomocí programu `psexec` [112] od *Sysinternals*, konkrétně prostřednictvím jeho přepínače `-s` [112]. Nastavení IL vyššího než je `0x4000` (*system*) se jeví jako velmi obtížné, protože Windows pravděpodobně neobsahuje účet s vyšším IL než *system*. V sekci 3.2.2.7 je IL *protected process* rozebírán se závěrem, že by teoreticky mohlo být možné proces s takovým IL spustit, ale Windows na tyto procesy klade poměrně restriktivní požadavky na ověřování a podepisování.

#### 3.2.5.2 Procesy

Při přiřazování *integrity levelu* procesům, lze využít *policy* `TOKEN_MANDATORY_NEW_PROCESS_MIN`, která zapříčiní, že vznikajícímu procesu je přiřazeno minimum z IL spouštějícího uživatele a IL nastaveného na souboru, ze kterého je proces spouštěn [91]. To znamená, že řízení IL vznikajícího procesu lze dosáhnout za pomocí programů diskutovaných v předchozí sekci 3.2.5.1.

Tato metoda funguje perfektně pro *integrity levely* v rozsahu `0x0` (*untrusted*) až `0x2000` (*medium*). Jakmile je ale překročena hranice IL *medium*, je pro spouštění programu nutné disponovat administrátorskými oprávněními resp. IL *high*. Pokud totiž dojde ke spuštění programu z běžného uživatelského účtu, je vlivem výše zmiňované *policy* vznikajícímu procesu přiřazen IL *medium*. Jako řešení by se mohlo jevit spustit program pomocí funkce *Run as administrator*. Nicméně, jak demonstrují experimenty v sekci A.2, v tomto případě `TOKEN_MANDATORY_NEW_PROCESS_MIN` nefunguje a výsledný proces získává IL *high*, a to i v případě, že byl na souboru nastaven IL, který je nižší než *high*, např. *medium plus*. V této situaci se totiž pravděpodobně aplikuje princip rozebíraný v sekci 3.2.3, podle kterého získá potomek procesu běžícího s IL *high* také IL *high*. Na totožné chování lze narazit i v rozpětí `0x3001` až `0x3fff` při používání programu `psexec` [112]. Dosažení *integrity levelu* vyššího než je *system* je, podobně jako při řízení IL souborů a adresářů, velmi obtížné.

Pro zaplnění těchto mezer vznikla jako součást této práce aplikace `il_proc`, která dokáže spustit libovolnou aplikaci s libovolným *integrity levelem* za předpokladu, že uživatel, který aplikaci spouští, má sám dostatečný IL. V tandemu s programem `psexec` [112] je za pomoci `il_proc` možné nastavovat IL na jakoukoliv hodnotu v rozmězí `0x0` (*untrusted*) až `0x4000` (*system*). Mechanismus jejího fungování je následující:

- Nejprve je pomocí `OpenProcessToken` získán handle k *access tokenu* procesu `il_proc`.
- Skrze `DuplicateTokenEx` je *access token* duplikován.
- Jako v `il_file` je funkcí `ConvertStringSidToSid` provedena konverze vstupního řetězce na SID.
- Funkcí `SetTokenInformation` je v duplikovaném *access tokenu* nastaven konvertovaný SID specifikující *integrity level*.
- Nakonec je pomocí funkce `CreateProcessAsUser` vytvořen nový proces s upraveným *access tokenem*.

V případě této aplikace není nutné, aby uživatel disponoval speciálními oprávněními, jako tomu bylo u `il_file`. Nicméně je potřeba, aby jeho *integrity level* byl aspoň takový, jako se snaží nastavit na vznikajícím procesu.

### 3.2.6 Chování v souborovém systému

V sérii experimentů A.1 je pozorováno chování *integrity levelů* v souborovém systému s cílem identifikovat možné vektory útoku, např. změnou obsahu souboru s vysokou integritou. Experimenty se zaměřují zejména na souborový systém *New Technology File System* (NTFS).

Úvodní experiment se zabývá kompilovanými a interpretovanými programy. Závěrem této části je, že rekompilace programu prostřednictvím Microsoft C/C++ kompilátoru [113] zachovává IL. To znamená, že obsah konkrétního programu psaného v C/C++ a pravděpodobně dalších kompilovaných programů lze nahradit, aniž by došlo ke ztrátě IL a *policies*. Podobný závěr lze učinit i z části experimentu s interpretovaným programem. V ní dochází k úspěšné úpravě programu psaného v jazyce Python [114] bez ztráty informací o *integrity levelu*.

Nicméně kvůli *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP`, která je ve výchozím nastavení přiřazena každému souboru a zamezuje *trustees* a procesům s nižším IL do souborů zapisovat, se nejedná o vektor, který by šel využít pro *privilege escalation*. V případě, že by tato *policy* nastavena nebyla, byla by tato metoda zneužitelná. I v případech, kdy je *policy* nastavena, hrozí nebezpečí od uživatelů se stejnou úrovní IL. Pokud jsou na daném souboru nesprávně nastavena přístupová práva, která umožní jinému uživateli než vlastníkovi do tohoto souboru zapisovat, nedokáže *Mandatory Integrity Control* poskytnout ochranu a soubor může být bez vědomí vlastníka nahrazen.

Druhá část experimentu se zaměřuje na operace v souborovém systému, jako je kopírování, přesouvání a mazání. Při kopírování bylo pozorováno, že *integrity level* není přenesen na výsledný soubor. Namísto toho je mu přiřazen IL v souladu s mechanismy popisovanými v sekci 3.2.3 a chování je totožné jako při vytváření nového souboru. Naproti tomu při přesouvání souborů v rámci stejného svazku je *integrity level* zachován včetně *policies*.

Překvapivé je, že soubory s vyšší integritou nejsou při základním nastavení chráněny proti přesouvání procesy a *trustees* s nižší integritou. K tomu, aby bylo ochrany dosaženo, je nutné nastavit *policies* `SYSTEM_MANDATORY_LABEL_NO_READ_UP` a `SYSTEM_MANDATORY_LABEL_NO_EXECUTE_UP`. Jelikož se jedná o poněkud nečekané chování a mohlo by být předpokládáno, že samotná *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP` bude dostačující pro zajištění takové úrovně ochrany, existuje zde jistá možnost pro pochybení a otevření vektoru pro útočníka.

U operace smazání souboru není ze zřejmých důvodů třeba diskutovat o tom, jak je naloženo s *integrity levelem*. Podobně jako při přesouvání souboru zde ale lze narazit na zvláštní chování, které spočívá v tom, že *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP` není dostačující pro to, aby zabránila smazání souboru s vyšší integritou procesem či *trustee* s nižší integritou. Ochrany je dosaženo až poté, co je na souboru nastavena *policy* `SYSTEM_MANDATORY_LABEL_NO_READ_UP`. Stejným způsobem jako v předchozím případě může toto chování být zneužitelné útočníkem za předpokladu, že vlastník souboru nesprávně nastavil přístupová práva.

V závěru skupiny experimentů A.1 je věnována pozornost dalším souborovým systémům, konkrétně *File Allocation Table* (FAT) a *Extensible File Allocation Table* (exFAT). Pro oba dokumentace uvádí, že nepodporují *security descriptory* [115]. To je ovšem zásadní problém, neboť uvnitř *security descriptoru* je uložen IL každého *securable objectu*. Z toho vyplývá, že ani FAT, ani exFAT neumožňují používat *integrity levely*. Na soubory uložené v těchto souborových systémech se tedy vztahují pravidla specifikovaná v sekci 3.2.3. Protože tyto soubory nemají *security descriptor*, jsou operačním systémem automaticky interpretovány, jako by měly IL *medium*.

V experimentech bylo identifikováno zvláštní chování, kdy byl proces s nižší integritou schopen zapsat do souboru s vyšší integritou. Zdá se, že na *securable objecty* uložené v souborových systémech FAT a exFAT se neaplikuje výchozí *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP`. Pokud uživatel není obeznámen s tím, že rodina souborových systému FAT nepodporuje řízení přístupu [116] ani *integrity levely*, může dojít k tomu, že do takového souborového systému uloží data, která budou při kompromitaci jeho systému volně dostupná útočníkovi. V extrémním případě by se mohlo stát, že nezkušený uživatel použije exFAT jako souborový systém pro svůj operační systém a úplně tak vyřadí z provozu bezpečnostní mechanismy jako *Mandatory Integrity Control* nebo *User Interface Privilege Isolation*. Takový scénář je ale vysoce nepravděpodobný.

## 3.3 Vyhodnocování přístupu

Při vyhodnocování přístupu k objektu předchází kontrola *integrity levelů* standardní kontrole DACL [82, 91, 94], protože je její vyhodnocení rychlejší než kompletní vyhodnocení DACL [82]. V rámci této kontroly rozhodne *Security Reference Monitor*, konkrétně funkce `SeAccessCheck` [82, 91], na základě *integrity levelů* a *policies*, jaké typy přístupů k danému objektu mohou být přiděleny skrze DACL [117].

Windows požaduje, aby každý proces, vlákno nebo jiný subjekt, který se chystá přistoupit k *securable objectu*, předem specifikoval, jaký typ přístupu požaduje. Následně zavolá *object manager*, komponenta, která řídí vytváření, mazání a přístupy k objektům ve Windows [82], funkci `SeAccessCheck`, které předá následující [82, 118]:

- informace o *securable objectu*, ke kterému je přistupováno, v podobě *security descriptoru*;
- informace o přistupujícím subjektu v podobě *security contextu*, který obsahuje *access token* [119];
- informace o tom, jaký typ přístupu subjekt požaduje.

Celou signaturu funkce `SeAccessCheck` lze vidět ve výpisu kódu 3.3. V případě, že je přístup vyhodnocen jako oprávněný, vrací funkce hodnotu `TRUE`, v opačných případech vrací `FALSE` [118].

**Výpis kódu 3.3** Funkční signatura funkce `SeAccessCheck` [118]

```
BOOLEAN SeAccessCheck(
        PSECURITY_DESCRIPTOR SecurityDescriptor,
        PSECURITY_SUBJECT_CONTEXT SubjectSecurityContext,
        BOOLEAN SubjectContextLocked,
        ACCESS_MASK DesiredAccess,
        ACCESS_MASK PreviouslyGrantedAccess,
        PPRIVILEGE_SET *Privileges,
        PGENERIC_MAPPING GenericMapping,
        KPROCESSOR_MODE AccessMode,
        PACCESS_MASK GrantedAccess,
        PNTSTATUS AccessStatus
);
```

Bezpečnostní subsystém Windows dovoluje aplikacím používat služby SRM skrze *user-mode* AuthZ API, které pro `SeAccessCheck` poskytuje ekvivalentní funkci `AccessCheck`. Tímto způsobem se mohou aplikace do bezpečnostního systému integrovat a využít jeho funkcionality [82]. `AccessCheck` má sice oproti svému *kernel-level* protějšku lehce pozměněné rozhraní, ale princip jejího fungování je totožný [120].

Část funkce `SeAccessCheck`, která se zabývá kontrolou *integrity levelů* a *policies*, se nazývá `MandatoryIntegrityCheck` a její pseudokód je vyobrazen ve výpisu kódu 3.4. Na vstupu tato funkce dostane *security descriptor* SO, ke kterému je přistupováno, a *access token* subjektu, který se o přístup pokouší. Výstupem jsou v tomto pseudokódu dva parametry, `result` a `allowed_access`. Parametr `result` nabývá hodnot `TRUE` resp. `FALSE` v případě, že je přístup povolen resp. odepřen. Druhý z dvojice parametrů, `allowed_access`, představuje množinu všech oprávnění, která mohou být subjektu přidělena. Následující DACL kontrola však může tato oprávnění dále upravovat. [117]

V první části funkce je zkontrolována *policy* v *access tokenu* přistupujícího subjektu. Pokud je nastavena na `TOKEN_MANDATORY_POLICY_OFF` nebo `TOKEN_MANDATORY_POLICY_NEW_PROCESS_MIN`,

**Výpis kódu 3.4** Pseudokód funkce `MandatoryIntegrityCheck` [117]

```
function MandatoryIntegrityCheck(access token, security descriptor)
    result ← FALSE                                              ▷ výsledek algoritmu
    allowed_access ← ∅                                          ▷ množina povolených přístupů
    token_policy ← TOKEN_MANDATORY_POLICY z access tokenu subjektu
    if token_policy = TOKEN_MANDATORY_POLICY_OFF
     or  token_policy = TOKEN_MANDATORY_POLICY_NEW_PROCESS_MIN then
        result ← TRUE
        add GENERIC_ALL to allowed_access
        return result, allowed_access
    end if
    ace ← SYSTEM_MANDATORY_LABEL_ACE ze security descriptoru SO
    ace_flags ← ace.ACE_HEADER.AceFlags
    ace_mask ← SYSTEM_MANDATORY_LABEL_NO_WRITE_UP               ▷ výchozí policy
    object_SID ← S-1-16-8192                                    ▷ výchozí hodnota SID – IL medium
    if INHERIT_ONLY_ACE ∉ ace_flags then
        ace_mask ← ace.Mask
        object_SID ← SID, na jehož první DWORD ukazuje ace.SidStart
    end if
    subject_SID ← SID ze struktury TOKEN_MANDATORY_LABEL access tokenu subjektu
    token_dominates ← FALSE
    if CompareSid(object_SID, subject_SID) = TRUE then
        token_dominates ← TRUE
    else if SidDominates(object_SID, subject_SID) = TRUE then
        token_dominates ← TRUE
    end if
    if token_policy = TOKEN_MANDATORY_NO_WRITE_UP then
        add GENERIC_READ, GENERIC_EXECUTE to allowed_access
        if token_dominates then
            add GENERIC_WRITE to allowed_access
        end if
    end if
    if !token_dominates then
        if ace_mask & SYSTEM_MANDATORY_LABEL_NO_READ_UP then
            remove GENERIC_READ from allowed_access
        end if
        if ace_mask & SYSTEM_MANDATORY_LABEL_NO_WRITE_UP then
            remove GENERIC_WRITE from allowed_access
        end if
        if ace_mask & SYSTEM_MANDATORY_LABEL_NO_EXECUTE_UP then
            remove GENERIC_EXECUTE from allowed_access
        end if
    end if
    token_privileges ← oprávnění z access tokenu subjektu
    if SeRelabelPrivilege ∈ token_privileges then
        add WRITE_OWNER to allowed_access
    end if
    return result, allowed_access
end function
```

neboli není nastavena na `TOKEN_MANDATORY_POLICY_NO_WRITE_UP`, je přístup rovnou udělen [117]. To proto, že absence této *policy* v *access tokenu* znamená, že přístup subjektu k objektům s vyšší integritou není nikterak omezen [121].

Následně dochází k načtení příznaků a masky ze `SYSTEM_MANDATORY_LABEL_ACE` *securable objectu*. Pokud hlavička této ACE obsahuje příznak `INHERIT_ONLY_ACE`, jsou aplikovány výchozí hodnoty pro SID a *policy*.

V další sekci pseudokódu jsou používány dvě funkce, `CompareSid` resp. `SidDominates`, jejichž pseudokód je vyobrazen ve výpisu kódu 3.5 resp. 3.6. `CompareSid` vrací `TRUE` v případě, že oba vstupní SID jsou totožné a `FALSE` v opačném případě. V případě, že vstupní SID totožné nejsou, je dále zavolána funkce `SidDominates`, která rozhodne, zda SID v prvním argumentu reprezentuje vyšší *integrity level* než SID v druhém argumentu a v takovém případě vrací hodnotu `TRUE`, v opačném případě vrací `FALSE` [122]. Využití těchto funkcí je poněkud pozoruhodné, jelikož operace, které provádí, se částečně překrývají. Nicméně za pozornost stojí, že `CompareSid` se věnuje i případům, kdy se první čísla SID označující verzi, někdy označovanou jako revizi, neshodují, na což `SidDominates` nepamatuje [117, 122].

**Výpis kódu 3.5** Pseudokód funkce `CompareSid` [117]

```
Require: sid_1 i sid_2 jsou SID udávající integrity level
  function CompareSid(sid_1, sid_2)
      if sid_1 a sid_2 mají různé verze (revize) then
          return FALSE
      end if
      subauth_count ← počet subautorit sid_1
      sid_length ← 8 + 4 · subauth_count
      return !memcmp(sid_1, sid_2, sid_length)
  end function
```

**Výpis kódu 3.6** Pseudokód funkce `SidDominates` [122]

```
Require: sid_1 i sid_2 jsou SID udávající integrity level
  function SidDominates(sid_1, sid_2)
      if sid_1 = sid_2 then
          return TRUE
      end if
      subauth_count_1 ← počet subautorit v sid_1
      subauth_count_2 ← počet subautorit v sid_2
      if subauth_count_2 > subauth_count_1 then
          return FALSE
      end if
      subauth_1 ← subautority v sid_1
      subauth_2 ← subautority v sid_2
      for i ∈ 0 . . . subauth_count_1 − 1 do
          if subauth_1[i] ≥ subauth_2[i] then
              return TRUE
          end if
      end for
      return FALSE
  end function
```

Poté dochází ke kontrole toho, zda je *policy* v *access tokenu* přistupujícího subjektu rovna `TOKEN_MANDATORY_POLICY_NO_WRITE_UP` [117] Zde se pravděpodobně nachází chyba. V případě,

že by se v *access tokenu* nacházela kombinace *policies* `TOKEN_MANDATORY_POLICY_NO_WRITE_UP` a `TOKEN_MANDATORY_POLICY_NEW_PROCESS_MIN`, která bývá v dokumentaci označována také jako `TOKEN_MANDATORY_POLICY_VALID_MASK` [121], nedojde k vykonání větve, ve které jsou do množiny `allowed_access` přidána patřičná oprávnění. Přestože by subjekt měl vyšší *integrity level* než *securable object*, byl by mu odepřen přístup, protože by mu nebyla udělena oprávnění, která mu právem náleží. Nicméně chyba je pravděpodobně pouze v pseudokódu a samotný algoritmus využívaný ve Windows je implementován správně, neboť, jak bylo řečeno v sekci 3.2.1.1, obě tyto *policies* jsou výchozími *policies* nastavovanými pro většinu procesů běžících v operačním systému [91] a chyba takového rozměru by se byla již projevila.

V případě, že *access token* přistupujícího subjektu obsahuje nižší *integrity level*, než je přiřazen *securable objectu*, ke kterému je přistupováno, jsou v následujícím bloku kódu postupně dle masky v `SYSTEM_MANDATORY_LABEL_ACE` odebírána jednotlivá přístupová práva [117]. Speciální pozornost je věnována `SeRelabelPrivilege`, které držiteli umožňuje měnit *integrity level* libovolného SO [123].

Kapitola 4

# *User Interface Privilege Isolation*

*User Inteface Privilege Isolation* patří do sady bezpečnostních opatření, které byly přidány do Windows s verzí Windows Vista [35]. UIPI je silně spjato s *Mandatory Integrity Control* už jen v tom ohledu, že se snaží v rámci *interactive desktop* izolovat procesy, které běží se zvýšenými oprávněními, od těch, které mají přiřazena běžná nebo nižší oprávnění [124], čímž brání *privilege escalation* útokům, jako jsou útoky typu *Shatter* [81].

Základ původní realizace útoku *Shatter* tkvěl v tom, že neprivilegovaný proces mohl privilegovanému procesu poslat *window message*, skrze kterou dokázal v privilegovaném procesu spustit libovolný kód. UIPI omezuje zprávy, které v takové situaci může neprivilegovaný proces privilegovanému procesu poslat [124, 91], a tím odstraňuje možné vektory útoku. Opačného směru, tedy případu, kdy privilegovaný proces posílá *window messages* neprivilegovanému procesu [91], či případu, kdy jeden proces posílá *window messages* druhému procesu se stejnou úrovní oprávnění [124], se UIPI netýká a odeslané *window messages* nejsou nikterak filtrovány.

Tato omezení jsou vymáhána subsystémem *USER*, který řídí grafické rozhraní a zobrazování oken [124, 91, 81]. K odlišení privilegovaných a neprivilegovaných procesů využívá *USER integrity levely* poskytované *Mandatory Integrity Control*. V momentu, kdy aplikace poprvé použije Windows *graphics device interface* (GDI), je *USER* inicializován a v rámci této inicializace provede volání do bezpečnostního subsystému Windows s cílem zjistit *integrity level* přiřazený do *access tokenu* daného procesu [91]. Tímto způsobem je procesu a případně jeho oknům přiřazena úroveň integrity, která je dále využívána v grafickém subsystému a po celou dobu běhu procesu zůstává neměnná [91]. To znamená, že proces sice může měnit svůj *integrity level*, ale v rámci grafického rozhraní bude stále mít ten IL, který měl nastaven při svém spuštění.

## 4.1 Zavedená omezení

V kontextu, kdy jeden proces běží s nižší integritou než druhý, nemůže proces s nižší integritou [124, 91, 81, 82]:

- provádět kontrolu platnosti *handle* okna procesu s vyšší integritou;
- posílat většinu *window messages* oknům procesu s vyšší integritou;
- používat *thread hooks* a *journal hooks* k připojení a monitorování procesu s vyšší integritou;
- injektovat DLL do procesu s vyšší integritou.

Dokumentace už nekonkretizuje, co je myšleno kontrolou platnosti *handle* okna procesu s vyšší integritou[1]. Prvotní dojem může nasvědčovat tomu, že se jedná o skupinu funkcí `IsWindow`, `IsWindowEnabled` a `IsWindowVisible`, nicméně, jak ukazuje experiment A.4, tyto funkce může proces s nižší integritou volat na *handle* okna s vyšší integritou. Tato fráze se dále ve Windows dokumentaci nevyskytuje, a proto není jasné, co jí bylo původně myšleno.

Ústředním polem působnosti UIPI jsou *window messages*. Jejich omezení je hlavním prostředkem, který vedl k zabránění útokům typu *Shatter*. UIPI blokuje veškeré zprávy s výjimkou následující skupiny zpráv [82, 35]:

- `WM_NULL`;
- `WM_MOVE`;
- `WM_SIZE`;
- `WM_GETTEXT`;
- `WM_GETTEXTLENGTH`;
- `WM_GETHOTKEY`;
- `WM_GETICON`;
- `WM_RENDERFORMAT`;
- `WM_DRAWCLIPBOARD`;
- `WM_CHANGECBCHAIN`;
- nedokumentovaná zpráva `0x313`;
- `WM_THEMECHANGED`;
- nedokumentovaná zpráva `0x31b`.

V následující sekci 4.4, která se věnuje chování filtru *window messages*, jsou tyto zprávy rozebírány ve větším detailu a porovnávány s výsledky experimentů na současné verzi Windows. V této sekci je také identifikováno několik zpráv, které na seznamu výše nejsou, přesto však projdou UIPI filtrem.

Další prostředek, který UIPI omezuje, jsou *hooks*, konkrétně jde o funkce `SetWindowsHookEx` a `SetWinEventHook` [81]. Skrze *hooky* lze totiž monitorovat *window messages* před jejich zpracováním ve *window procedure* či po něm nebo také *window messages*, které jsou daným procesem odesílány [125, 81]. Dále umožňuje sledovat stisky kláves, pohyby myší nebo zápisy do systémového logu [125, 82]. Conover [81] dokonce zmiňuje, že skrze DLL, které je do cílového procesu injektováno jako výsledek volání těchto funkcí, bylo možné zneužít pro spuštění libovolného kódu v tomto procesu. S tím souvisí i poslední bod omezení zavedených UIPI, který zajišťuje, že procesy s nižší integritou nemohou injektovat DLL do procesů s vyšší integritou.

Mezi objekty *USER* subsystému, které navzdory UIPI stále zůstávají sdílené mezi procesy s různou integritou, patří [124, 91]:

- okno plochy, což je okno, na kterém se vykreslují okna všech procesů;
- paměť pouze pro čtení spadající do haldy plochy;
- globální tabulka atomů;

[1] V originále formulováno jako „*Perform a window handle validation of a process running with higher rights.*“ [91]

- schránka.

Dále UIPI neřídí vykreslování na ploše, neboť ani *USER*, ani GDI nepodporují řízení přístupu k prostorům, na které se mohou okna vykreslovat. Jinými slovy je možné, aby aplikace s nižší integritou vykreslovala okna přes oblast, kterou pro vykreslování používá aplikace s vyšší integritou. [124, 91]

## 4.2 *UIAccess*

Omezení stanovená UIPI mohou být pro některé aplikace, které ze své podstaty musí komunikovat či ovlivňovat ostatní procesy, nežádoucí. Jedná se zejména aplikace pro zajištění přístupnosti operačního systému a jeho ovládání [91]. Typickým příkladem jsou aplikace, jejichž prostřednictvím ovládá uživatel jiné aplikace [91], jako *on-screen keyboard* (`osk.exe`) [82]. Pokud je prostřednictvím *on-screen keyboard* ovládána aplikace s IL *medium*, tak za předpokladu, že samotná *on-screen keyboard* běží se standardním IL *medium*, bude vše probíhat bez problému [91]. Nicméně pokud by bylo nutné pomocí *on-screen keyboard* ovládat aplikaci s vyšším IL než má samotný `osk.exe` proces, došlo by k selhání, neboť mechanismus *on-screen keyboard* pravděpodobně spoléhá na možnost posílat ovládanému procesu *window messages* [91].

Jako řešení by se mohlo nabízet danou aplikaci spustit pro každý IL zvlášť. Pokud by tímto způsobem byly pokryty všechny úrovně IL, bylo by možné skrze tuto aplikaci ovládat libovolnou jinou aplikaci [82]. Microsoft ale přichází s jednodušším řešením v podobě *UIAccess*. *UIAccess* je bezpečnostní atribut, o nějž si může aplikace zažádat při spuštění prostřednictvím svého aplikačního manifestu [91].

V kontextu Windows aplikací je aplikační manifest soubor ve formátu *Extensible Markup Language* (XML) obsahující metadata o aplikaci, jako je kompatibilita nebo závislosti. Navíc lze jeho prostřednictvím žádat o spuštění s administrátorskými oprávněními nebo speciálními atributy, jako je *UIAccess* [126]. Úryvek aplikačního manifestu, který značí, že aplikace žádá o přidělení *UIAccess* atributu, je zobrazen ve výpisu kódu 4.1.

**Výpis kódu 4.1** Úryvek aplikačního manifestu s žádostí o *UIAccess* [91]

```
<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
  <security>
    <requestedPrivileges>
    <requestedExecutionLevel
      level="asInvoker"
      UIAccess="true" />
    </requestedPrivileges>
  </security>
</trustInfo>
```

Aby se zabránilo neoprávněnému získání atributu *UIAccess*, musí aplikace žádající o jeho přidělení splňovat následující podmínky [91]:

- aplikace musí být opatřena digitálním podpisem, který je možné ověřit pomocí některého z digitálních certifikátů nainstalovaných v úložišti *Trusted Root Certification Authorities*;
- aplikace se musí nacházet v adresářovém stromu *Program Files* (`%ProgramFiles%`) nebo Windows (`%WinDir%`).

Dokumentace dále specifikuje, že umístění aplikace musí patřit mezi ta umístění, do kterých smí zapisovat pouze administrátor. To znamená, že pro `%WinDir%` existují jisté výjimky, neboť v jejím

adresářovém stromě se nachází několik umístění, do kterých mohou zapisovat i běžní uživatelé. Omezení na umístění programu lze řídit prostřednictvím nastavení dostupných v *Local Security Policy*. [91]

V případě, že aplikace ve svém manifestu požádá o přídělení atributu *UIAccess* a splní všechny požadavky vytyčené výše, je spuštěna se speciálními oprávněními, které jí umožňují obejít některá omezení stanovená UIPI. Taková aplikace smí [91]:

- určit, které okno se bude zobrazovat v popředí;
- posílat vstupy v podobě stisků kláves, pohybů myší a stisků tlačítek na myši [127] libovolnému oknu jiné aplikace prostřednictvím funkce `SendInput`;
- číst vstupy libovolného okna jiné aplikace pomocí *hooků*, funkcí pro práci s *raw input*, `GetKeyState`, `GetAsyncKeyState` a `GetKeyboardInput`;
- používat *journal hooks* k připojení a monitorování libovolných procesů;
- napojit se na frontu se vstupy libovolného vlákna pomocí funkce `AttachThreadInput`.

Jak bylo již zmíněno v sekci 3.2.2.4 o IL *medium plus*, procesům běžícím s *UIAccess* je přiřazen IL, který je inkrementován o hodnotu `0x10`. To znamená, že proces, který by standardně běžel s IL *medium*, tedy tedy RID `0x2000`, bude mít přiřazen RID `0x2010`. [91]

## 4.3 Interakce s filtrem *window messages*

Funkce `ChangeWindowMessageFilter` a `ChangeWindowMessageFilterEx` umožňují přímo modifikovat *User Interface Privilege Isolation* filtr, a to tak, že pro konkrétní zprávu lze definovat, zda jí má být povoleno skrz filtr projít nebo naopak zda má být blokována. Toho je docíleno pomocí *flagů* `MSGFLT_ADD` a `MSGFLT_REMOVE` u `ChangeWindowMessageFilter` resp. `MSGFLT_ALLOW` a `MSGFLT_DISALLOW` u `ChangeWindowMessageFilterEx`. V rozporu s intuicí ale může být, že přidáním zprávy do filtru (pomocí `MSGFLT_ADD`) jí umožníme skrze filtr procházet [128]. Tento fakt je sice důkladně zdokumentován v oficiální příručce, nicméně stále otevírá prostor pro pochybení na straně programátora. V případě špatně nastaveného filtru se aplikace stává náchylnou na útoky typu *Shatter*.

V současné době se od používání funkce `ChangeWindowMessageFilter` upouští a Microsoft dokonce uvádí, že v dalších verzích Windows nemusí být podporována [128]. Oproti svému předchůdci poskytuje `ChangeWindowMessageFilterEx` několik vylepšení. Úpravy filtru aplikuje cíleně na konkrétní okno, které lze určit pomocí jeho *handle*, namísto na celý proces, umožňuje resetovat pravidlo pro konkrétní zprávu na výchozí hodnotu nebo také získat detailnější informace o výsledku volání funkce prostřednictvím struktury `CHANGEFILTERSTRUCT` [129].

Důležité je zmínit, že obě funkce mají svá omezené. Dokumentace uvádí, že některé zprávy nižší než `WM_USER` budou filtrem propuštěny bez ohledu na to, jak je programátorem v aplikaci nastaven. Na pokus o blokaci takové zprávy odpoví funkce `ChangeWindowMessageFilter` návratovou hodnotou `TRUE` jako při úspěchu, ale zpráva zablokována nebude [128]. Dalším omezením je, že ani jedna z funkcí nemůže být používána procesy, které mají *integrity level low* nebo nižší [128, 129].

V experimentu A.5 jsou *window messages* klasifikovány dle jejich stavu do několika kategorií, jejichž pojmenování jsou místy používána i v následujícím rozboru. Jejich definice lze najít v úvodu experimentu A.5. Podobně se v následujícím textu mohou vyskytovat odkazy na chybové kódy funkcí `SendMessage` a `PostMessage`. Přehled všech takových kódů, které byly v experimentech zaznamenány, a jejich významů je k dispozici v tabulce A.2.

## 4.4 Chování filtru *window messages*

Pro testování *window messages* vznikly jako součást této práce dva programy, `sender.exe` a `receiver.exe` (oba jsou dostupné na přiloženém médiu). Program `sender.exe` implementuje funkcionalitu potřebnou pro posílání libovolných *window messages*. V experimentech je nejčastěji používán tak, že postupně prochází všechny *window messages* v rozsahu `0x0` až `0xffff` a snaží se je poslat na cílové okno, ať už pomocí funkce `SendMessage` nebo `PostMessage`.

Jeho protějšek `receiver.exe`, jak již název napovídá, slouží k přijímání zpráv. Tato aplikace sestává z *message loop* ve funkci `wWinMain`, ze které jsou *window messages* předávány do `WindowProc`. V obou úrovních jsou přijaté *window messsages* vypisovány na výstup včetně jejich parametrů, identifikátorů a názvů. Detailnější dokumentaci obou aplikací lze nalézt na přiloženém médiu.

Pomocí těchto aplikací je v experimentech v sekci A.5 pozorováno chování filtru *window messages* jak v situaci, kdy UIPI není aktivní, tak v situaci, kdy je. Protože testovací aplikace mají svá omezení, je každá sporná zpráva dohledána v dokumentaci, existuje-li, a poté je manuálně testována dle postupu popsaného v sekci A.5.3. Zprávy, pro které dokumentace neexistuje, jsou v tabulkách označeny pouze hexadecimální číslicí namísto názvu a proces manuálního testování je pro ně značně omezen.

### 4.4.1 Bez aktivovaného *User Interface Privilege Isolation* (`SendMessage`)

V případě, že jsou obě aplikace spuštěny se stejným *integrity levelem*, není filtrování *window messages* UIPI aktivní. Při použití funkce `SendMessage` *window messages* bylo mezi všemi odeslanými *window messages* detekováno 160 zpráv, které nebyly přijaty `WindowProc` programem `receiver.exe`; jejich přehled se nachází v tabulce A.3.

#### 4.4.1.1 Blokované *window messages*

Za blokované zprávy jsou považovány všechny zprávy, pro které při manuálním testování vrátila funkce `GetLastError` chybový kód `0x5` („*Access is denied.*“ [33]). Pokud by testovací aplikace běžely na různých *integrity levelech*, byl by tento chybový kód indikátorem toho, že zpráva byla blokována UIPI [31, 32]. Protože však v aktuálním kontextu není UIPI aktivní, musí být zprávy blokovány z jiných důvodů.

U zpráv `WM_CREATE` [130], `WM_NCCREATE` [131] a `WM_INITDIALOG` [132] dokumentace hovoří o tom, že tyto zprávy jsou oknům resp. dialogovým oknům doručovány bezprostředně po jejich vytvoření příslušnou funkcí (např. u `WM_CREATE` jde o funkce `CreateWindow` a `CreateWindowEx`). Můžeme tedy předpokládat, že běžné programy tyto zprávy nemohou posílat a jsou generovány pouze příslušnými funkcemi.

V případě zprávy `WM_DEVMODECHANGE` dokumentace [133] tvrdí, že tuto zprávu není možné poslat konkrétnímu oknu skrze `SendMessage` a na místo ní má být použita `SendMessageTimeout` v režimu *broadcast*. Nicméně i s touto funkcí měla `WM_DEVMODECHANGE` stále chybový kód `0x5`. Je možné, že pro použití této zprávy jsou nutná vyšší oprávnění, dokumentace se však o takovém omezení nezmiňuje. Odlišná situace nastává u *window message* `WM_NOTIFY`, u níž je v dokumentaci jednoznačně stanoveno, že od Windows Vista nesmí být posílána mezi procesy [134].

Do kategorie blokovaných zpráva lze zařadit i několik zpráv, které nebyly při odeslání pomocí `SendMessage` doručeny a při použití `PostMessage` měly chybový kód `0x5`. Jedinou zprávou tohoto typu, která je zdokumentovaná, je zpráva `WM_TOOLTIPDISMISS`. Tato zpráva je posílána všem oknům, která si její příjem vyžádají pomocí funkce `RegisterForTooltipDismissNotification`, a indikuje změny stavu lišty nástrojů [135]. Na základě dokumentace bychom mohli předpokládat, že průchod `WM_TOOLTIPDISMISS` UIPI filtrem je podmíněn předchozím voláním zmiňované

funkce. Možná je i situace podobná zprávám `WM_CREATE`, `WM_NCCREATE` a dalším, které jsou pravděpodobně generovány pouze operačním systémem skrze určité Windows API funkce.

#### 4.4.1.2 Nezprovozněné *window messages*

Některé zprávy nebylo možné v žádné ze zkoušených konfigurací doručit. U zpráv `WM_GESTURE` resp. `WM_TOUCH` je tomu tak z toho důvodu, že jako `lParam` požadují tyto *window messages handle* typu `HGESTUREINFO` resp. `HTOUCHINPUT`, které jsou ve Windows poskytovány pouze operačním systémem. Je tedy zřejmé, že tyto *window messages* nejsou určeny pro posílání mezi procesy ale pouze pro příjem informací od operačního systému. Pokusy o odeslání `WM_GESTURE` a `WM_TOUCH` typicky končily chybovým kódem `0x57` „*The parameter is incorrect,*“ [33].

K podobnému závěru můžeme dojít i u zpráv týkajících se ukazovacích zařízení, tedy zpráv s prefixem `WM_NCPOINTER`, `WM_POINTER` nebo `DM_POINTER`. Některé z nich jako `DM_POINTERHITTEST` a `WM_POINTERROUTEDTO` podle dokumentace nevyžadují speciální obsah parametrů [136, 137], čímž klesá pravděpodobnost, že by byly při automatizovaném či manuálním testovány využity nesprávně. Kromě *window messages* `WM_POINTERDEVICECHANGE`, `WM_POINTERDEVICEINRANGE` a `WM_POINTERDEVICEOUTOFRANGE`, které bylo možné doručit pomocí `PostMessage`, se nepodařilo žádnou další zprávu přijmout na `receiver.exe`, přestože při odesílání neindikovala funkce `GetLastError` žádnou chybu.

Ani u zpráv `EM_GETLINE` a `WM_MDICREATE` se nepodařilo najít vhodnou konfiguraci parametrů. `EM_GETLINE` je spjatá s *multiline* ovládacím prvkem [138], přesto však při jejím použití na ovládací prvky ve vestavěných aplikacích Windows a sledování pomocí Spy++ [139] nebyla doručena. Podobná situace s týká i `WM_MDICREATE`, která se používá k ovládání *multiple-document interface* (MDI) [140].

#### 4.4.1.3 *Window messages* doručené `PostMessage`

Speciální skupinu zpráv tvoří zprávy, které, aniž by indikovaly chybu, nebyly doručeny pomocí `SendMessage` a při použití `PostMessage` se jejich doručení zdařilo. Do této skupiny se řadí zprávy spjaté s *Dynamic Data Exchange* (DDE), které lze poznat tak, že jsou prefixovány `WM_DDE`. U každé z nich s výjimkou `WM_DDE_FIRST`/`WM_DDE_INITIATE` je v dokumentaci [141, 142, 143, 144, 145, 146, 147, 148] uvedeno, že pro jejich odeslání musí být použita funkce `PostMessage`. Zpráva `WM_DDE_FIRST`/`WM_DDE_INITIATE` však není jedinou *window message*, kterou se zdařilo odeslat pomocí `SendMessage`, úspěšného doručení bylo dosaženo i pro zprávu `WM_DDE_ACK`.

#### 4.4.1.4 Neodeslané *window messages*

V tabulce A.4 se nachází výpis 34 zpráv, které byly přijaty `receiver.exe`, aniž by byly odeslány `sender.exe`. Část těchto zpráv můžeme zavrhnout jako irelevantní a připsat je dalšímu dění na počítači v průběhu experimentu. Typickým příkladem je *window message* `WM_GETICON`, kterou program obdrží i při pouhém přejetí přes jeho ikonu na spodní liště Windows *graphical user interface* (GUI).

Nicméně mezi nevyžádanými zprávami je i několik úkazů situace, kdy jedna *window message* způsobí přijetí několika dalších. Mezi ně patří např. `WM_IME_NOTIFY`, `WM_DWMCOMPOSITIONCHANGED` nebo i nedokumentovaná zpráva `0x313`. Všechny tyto případy a další jsou popsány v tabulce A.4. U některých z nich nastává pozoruhodná situace, kdy zprávy generované navíc, jsou přijaty pouze při prvním doručení generující *window message* a při dalších už tento jev nenastává.

#### 4.4.1.5 Ostatní *window messages*

Zbývající zprávy z analyzovaných 160 zpráv se podařilo pomocí manuálního testování a úprav popsaných v sekci A.5.3 doručit a nebyly předmětem dalšího zkoumání. Pro většinu z nich postačila obecná úprava, nicméně některé zprávy vyžadovaly úpravu komplexnější.

Za zmínku stojí, že tímto způsobem se zdařilo doručit i několik nedokumentovaných zpráv. To vypovídá o tom, že tyto zprávy pravděpodobně mají své využití, které je buď interní pro operační systém, nebo pro něj jednoduše chybí dokumentace.

### 4.4.2 Bez aktivovaného *User Interface Privilege Isolation* (`PostMessage`)

Podobně jako experiment, jehož výsledky jsou analyzovány v předchozí sekci 4.4.1, se i experiment A.5.1.2 věnuje sledování komunikace pomocí *window messges* bez aktivního UIPI filtru, ale namísto `SendMessage` používá funkci `PostMessage`. V tomto případě bylo nalezeno 169 zpráv, popsaných v tabulce A.5, které nebyly doručeny do *message loop* ve `wWinMain`.

#### 4.4.2.1 Blokované *window messages*

Skupina blokovaných zpráv obsahuje některé zprávy, které byly rozebírány již v předchozí sekci 4.4.1, jako jsou `WM_NOTIFY` a `WM_TOOLTIPDISMISS`. To, že zpráva `WM_TOOLTIPDISMISS` je při použití `PostMessage` explicitně blokována, podporuje dřívější teorii o tom, že je typicky generována pouze operačním systémem a není určena pro manuální posílání mezi procesy.

Speciální podskupinu blokovaných *window messages* tvoří zprávy `0x2ed`, `0x2ee`, `0x348` a `0x349`. Jedná se sice o nedokumentované zprávy, takže o důvodu jejich blokace je možné pouze spekulovat, nicméně pozoruhodné jsou tím, že při odeslání pomocí `PostMessage` jsou blokovány s chybovým kódem `0x5`, ale při použití `SendMessage` jsou úspěšně doručeny. To hovoří o tom, že pravděpodobně mají ve Windows využití, které buď není součástí veřejného API, nebo není zdokumentováno.

#### 4.4.2.2 Nezprovozněné *window messages*

Stejně jako experiment A.5.1.1 zahrnuje i experiment A.5.1.2 několik zpráv, pro které se v průběhu testování nezdařilo nalézt takovou konfiguraci, která by umožnila jejich doručení. Podobně jako v předchozí skupině se i zde nachází již známé zprávy `WM_GESTURE` a `WM_TOUCH`. Z podobných důvodů, které jsou rozebírány v sekci 4.4.1, nebylo možné doručit ani zprávu `WM_DROPFILES`. Ta v jednom ze svých parametrů vyžaduje *handle* typu `HDROP` [149], který je pravděpodobně možné získat pouze od operačního systému, jenž tyto zprávy generuje. Při testování vracelo její volání konzistentně chybový kód `0x6` – „*The handle is invalid*“ [33]. Tím se odlišuje od zmiňovaných zpráv `WM_GESTURE` a `WM_TOUCH`, které reagují chybovým kódem `0x57`. Nicméně podobně jako u těchto zpráv je pravděpodobné, že `WM_DROPFILES` slouží pouze pro přijímání informací od operačního systému a není určena pro komunikaci napříč procesy.

Velkou část skupiny těchto *window messages* tvoří zprávy s chybovým kódem `0x3ea`, kterému odpovídá vysvětlivka „*The window cannot act on the sent message*“ a konstanta s názvem `ERROR_INVALID_MESSAGE` [150]. Do této skupiny patří i zprávy prefixované `WM_NCPOINTER`, `WM_POINTER` a `DM_POINTER` s výjimkou `WM_POINTERDEVICECHANGE`, `WM_POINTERDEVICEINRANGE` a `WM_POINTERDEVICEOUTOFRANGE`. Vzhledem k tomu, že ani v předchozím experimentu s funkcí `SendMessage` nebylo možné zajistit úspěšné doručení těchto zpráv, mohli bychom se uchýlit k závěru, že tyto zprávy jsou generovány pouze operačním systémem, tak jako jsme spekulovali právě v předchozím experimentu. Nicméně vzhledem k tomu, že oproti předchozímu experimentu máme nyní navíc k dispozici chybový kód, nabízí se ještě to vysvětlení, že aby bylo možné tyto *window messages* úspěšně přijímat, musí jejich příjemce zavolat určitou funkci Windows API, která spustí jejich zpracování.

Další podskupina nedoručených *window messages* sestává ze zpráv spojených s DDE. Jak již bylo řečeno v sekci 4.4.1, dokumentace specifikuje, že všechny tyto zprávy kromě `WM_DDE_FIRST`, označované také jako `WM_DDE_INITIATE`, mají být posílány skrze `PostMessage`. Nicméně mimo zprávu `WM_DDE_TERMINATE`, která prošla již automatizovaným testováním s parametry `0x0`, `0x0`,

nebylo možné žádnou zprávu doručit. Při automatizovaném testování pomocí `sender.exe` nastavila každá z těchto zpráv chybový kód na `0x578` („*Invalid window handle*“ [151]). To pravděpodobně proto, že ve svém `wParam` většinou vyžadují *handle* okna. Skrze manuální testování bylo možné všechny zprávy uvést do stavu, kdy nekončily chybou (`GetLastError` zůstala na `0x0`), ale přesto nebyly zprávy doručeny do `reciever.exe`. Pravděpodobně se nejedná o chybu v nastavení parametrů, jelikož zpráva `WM_DDE_UNADVISE` nevyžaduje komplexní nastavení [143], ale problém je spíše v tom, že by cílová aplikace měla provozovat aktivní DDE server, který je schopen tyto *window messages* zpracovávat.

Nejčetnější skupinu tvoří zprávy, pro které vrátila `GetLastError` hodnotu `0x487` odpovídající vysvětlivce „*The message can be used only with synchronous operations*“ [150]. Z ní je zřejmé, že tyto *window messages* nemohou být použity společně s funkcí `PostMessage`. Jak je rozebráno v sekci 1.4, funkce `PostMessage` nečeká, až cílový proces dokončí zpracování *window message* a ihned se z volání navrátí [32]. Odeslání zprávy pomocí `PostMessage` lze tedy chápat jako asynchronní operaci. Naproti tomu funkce `SendMessage` je synchronní, neboť z volání této funkce dojde k navrácení až poté, co cílové okno danou zprávu zpracuje [31].

### 4.4.3 S aktivovaným *User Interface Privilege Isolation* (`SendMessage`)

Pokud jsou testovací aplikace spuštěny na různých *integrity levelech*, dojde k aktivaci UIPI filtru a aplikaci principů diskutovaných v sekci 4.1. Výsledkem je, že při použití `SendMessage` je do `WindowProc` doručeno pouhých 16 *window messages* uvedených v tabulce 4.1. Kromě nich bylo detekováno dalších 107 zpráv, které přestože nebyly blokovány, nebyly ani doručeny na `receiver.exe`. Přehled obou dvou skupin zpráv včetně výsledků manuálního testování je k dispozici v tabulce A.7, která je součástí experimentu A.5.2.1.

#### 4.4.3.1 Blokované *window messages*

Mezi blokované zprávy patří naprostá většina odeslaných zpráv. Konkrétně se jedná o 65 411 *window messages*, ke kterým je ještě nutné přičíst 41 *window messages*, jež byly identifikovány jako blokované v rámci dodatečného testování.

Podobně jako u experimentu A.5.1.1 lze i v tomto případě mezi blokované zprávy teoreticky řadit ty, které nebyly pomocí `SendMessage` doručeny, aniž by indikovaly blokaci, ale při použití `PostMessage` již blokovány jednoznačně byly. Do této kategorie překvapivě patří např. zpráva `WM_NCCREATE`, která v experimentu A.5.1.1 bez UIPI byla explicitně blokována. V tomto případě však k explicitní blokaci dochází až při použití `PostMessage` a u `SendMessage` není indikována.

Dále skupina *window messages* blokovaných `PostMessage` zahrnuje několik *window messages*, které v experimentu A.5.1.1 nebyly zprovozněny, jako je např. `WM_COPYGLOBALDATA`, `EM_GETLINE` nebo `WM_MDICREATE`. To značí, že u některých *window messages* se filtrovací mechanismus UIPI pravděpodobně aplikuje ještě dříve, než provedou kontrolu svých parametrů.

#### 4.4.3.2 Nezprovozněné *window messages*

Podobně jako ve všech ostatních experimentech, obsahuje i experiment A.5.2.1 řadu zpráv, které se nepodařilo přes veškerou snahu doručit nebo alespoň zjistit, proč není jejich doručení možné. V této kategorii se opakuje několik již dobře známých zpráv, jako jsou `WM_GESTURE`, `WM_TOUCH` či vybrané zprávy s prefixy `WM_NCPOINTER`, `WM_POINTER` a `DM_POINTER`. Analýza těchto zpráv a odůvodnění jejich nefunkčnosti v experimentech je detailně rozebíráno v sekcích 4.4.1.2 a 4.4.2.2.

#### 4.4.3.3 Neodeslané *window messages*

I v experimentu A.5.2.1 bylo programem `receiver.exe` přijato celkem 11 zpráv, které nebyly programem `sender.exe` odeslány. Všechny tyto zprávy jsou shrnuty v tabulce A.8, kde je také objasněn důvod, proč došlo k jejich vygenerování. Této problematice se již věnovala sekce 4.4.1.4. Podobně jako dříve patří mezi tyto zprávy jak zprávy způsobené vedlejšími jevy na zařízení, na kterém experiment probíhal, a jsou tak irelevantní pro výsledky experimentu, tak zprávy, které jsou generovány jinými zprávami.

#### 4.4.3.4 Doručené *window messages*

Nejdůležitějším výstupem experimentu A.5.2.1 je seznam *window messages*, které byly i přes aktivované UIPI filtrování doručeny na cílovou aplikaci. Tyto zprávy jsou shrnuty v tabulce 4.1.

**Tabulka 4.1** Doručené *window messages* (`SendMessage` s UIPI)

| Zpráva | Návratová hodnota | Chybový kód |
|---|---|---|
| WM_NULL | 0x0 | 0x0 |
| WM_MOVE | 0x0 | 0x0 |
| WM_SIZE | 0x0 | 0x0 |
| WM_GETTEXT | 0x0 | 0x0 |
| WM_GETTEXTLENGTH | 0xd | 0x0 |
| WM_GETHOTKEY | 0x0 | 0x0 |
| WM_GETICON | 0x0 | 0x0 |
| WM_RENDERFORMAT | 0x0 | 0x0 |
| WM_DRAWCLIPBOARD | 0x0 | 0x0 |
| WM_CHANGECBCHAIN | 0x0 | 0x0 |
| 0x313 | 0x0 | 0x0 |
| WM_THEMECHANGED | 0x0 | 0x0 |
| 0x31b | 0x1 | 0x0 |
| WM_DWMNCRENDERINGCHANGED | 0x0 | 0x0 |
| 0xc0a1 | 0x0 | 0x0 |
| 0xc0a2 | 0x0 | 0x0 |

Při porovnání obsahu této tabulky se seznamem zpráv, které mají být výjimkami ve filtru UIPI, v sekci 4.1, lze pozorovat, že tabulka 4.1 obsahuje jednu dokumentovanou a dvě nedokumentované zprávy navíc. Těmito zprávami jsou `WM_DWMNCRENDERINGCHANGED`, `0xc0a1` a `0xc0a2`. Rozborem zpráv, které prošly UIPI filtrem, se dále zabývá sekce 4.4.6.

### 4.4.4 S aktivovaným *User Interface Privilege Isolation* (`PostMessage`)

Poslední experiment, A.5.2.2, ze série experimentů věnujících se UIPI filtru *window messages* se stejně jako experiment A.5.2.1 zabývá případem, ve kterém je filtr aktivní. Namísto funkce `SendMessage` však pro odesílání *window messages* používá funkci `PostMessage`. V tomto experimentu bylo identifikováno o dvě méně *window messages*, tedy 14, které byly úspěšně doručeny do `wWinMain` programu `receiver.exe`. U dalších 29 *window messages* uvedla funkce `PostMessage` jiný chybový kód než `0x5`, který by signalizoval blokaci UIPI. Přehled všech zpráv, ať už těch, které byly doručeny, tak těch, které nebyly explicitně blokovány, se nachází v tabulce A.9. Na rozdíl od experimentu A.5.2.1 nebyly v tomto případě do `wWinMain` doručeny žádné zprávy, které by program `sender.exe` neodeslal.

#### 4.4.4.1 Blokované *window messages*

Dle očekávání byla kvůli UIPI odfiltrována drtivá většina *window messages*, konkrétně 65 491. Dodatečné testování tentokrát neidentifikovalo žádné *window messages*, které by se daly považovat za dodatečně blokované. Pro diagnostiku filtru se tedy jeví přímočařejší použít funkci `PostMessage`, neboť oproti experimentu A.5.2.1 byly chybové kódy `0x6`, `0x57` a `0x578`, které indikují špatné nastavení parametrů, hlášeny pouze ve třech případech. To potvrzují zejména zprávy spojené s DDE, které se většinou v předchozích experimentech nepodařilo zprovoznit a typicky indikovaly chybu `0x578`, ale v tomto experimentu byly dle jejich chybového kódu jednoznačně odfiltrovány UIPI.

Můžeme si povšimnout, že *window messages*, které v experimentu A.5.1.2 bez aktivovaného UIPI hlásily, že jsou použitelné pouze s funkcí `SendMessage`, v tomto experimentu žádnou chybu neuvádějí a namísto toho explicitně indikují blokaci. Chybový kód `0x5` má tedy precedenci před chybovým kódem `0x487`.

#### 4.4.4.2 Nezprovozněné *window messages*

Jak bylo nastíněno výše, chybové kódy indikující chybu v parametrech se v experimentu A.5.2.2 vyskytovaly pouze minimálně. O jejich zastoupení se postaraly dobře známé *window messages* `WM_GESTURE`, `WM_DROPFILES` a `WM_TOUCH`. Důvody, proč tyto zprávy není možné v rámci experimentů zprovoznit, byly již pokryty v sekcích 4.4.1.2 a 4.4.2.2.

Podobně jako u experimentu A.5.1.2, který také využívá funkci `PostMessage`, se i v tomto experimentu objevila celá řada zpráv s kryptickým chybovým kódem `0x3ea`. Jedná se o zprávy s prefixy `WM_NCPOINTER`, `WM_POINTER` a `DM_POINTER` s výjimkou zpráv `WM_POINTERDEVICECHANGE`, `WM_POINTERDEVICEINRANGE` a `WM_POINTERDEVICEOUTOFRANGE`, které jsou explicitně blokovány. Předchozí zkušenost, kdy chybový kód `0x5` měl prioritu před chybovým kódem `0x487` by nás mohla vést k závěru, že tyto zprávy nejsou ve skutečnosti filtrovány UIPI a při správném použití by bylo jejich doručení povoleno. Nicméně Barbosa [35] a Russinovich [82], dva ze zdrojů četně citovaných v rešeršní části této práce, jednoznačně tvrdí, že tyto zprávy nepatří na seznam zpráv, které mohou UIPI filtrem procházet. Ostatně i v datech experimentu A.5.2.1 existuje několik případů dodatečně blokovaných zpráv, tedy zpráv, které nejprve indikovaly chybový kód jiný než `0x5`, ale při manuálním testování se ukázalo, že jsou ve skutečnosti blokovány UIPI. Jako pravděpodobnější vysvětlení se nabízí teorie formulovaná v sekci 4.4.2.2, podle které jsou tyto zprávy generovány pouze operačním systémem a úkolem programů je je pouze přijímat a nikoliv posílat. Případně ještě teorie, která tvrdí, že pro úspěšné přijetí těchto *window messages* musí příjemce zavolat jistou funkci Windows API, čímž spustí jejich zpracování.

#### 4.4.4.3 Doručené *window messages*

Počet zpráv úspěšně přijatých ve `wWinMain` programu `receiver.exe` se, jak naznačil úvod této sekce, oproti experimentu A.5.2.1 s funkcí `SendMessage` snížil o dva. Na vině je jev identifikovaný již v experimentu A.5.1.2 a detailně popsaný v sekci 4.4.2.2, kdy některé *window messages* nesmí být použity s funkci `PostMessage` a namísto toho musí pro jejich odeslání být použita funkce `SendMessage`. Kvůli němu z množiny doručených zpráv popsané tabulkou 4.2 ubyly zprávy `WM_GETTEXT` a `WM_GETTEXTLENGTH`.

Pokud tabulku 4.2 porovnáme se seznamem výjimek z filtru *window messages*, můžeme si povšimnout, že, konzistentně s experimentem A.5.2.1, obsahuje navíc dokumentovanou zprávu `WM_DWMNCRENDERINGCHANGED` a dvě nedokumentované. Rozchází se ale v tom, o jaké nedokumentované zprávy se jedná.

**Tabulka 4.2** Doručené *window messages* (`PostMessage` s UIPI)

| Zpráva | Návratová hodnota | Chybový kód |
|---|---|---|
| WM_NULL | 0x0 | 0x0 |
| WM_MOVE | 0x0 | 0x0 |
| WM_SIZE | 0x0 | 0x0 |
| WM_GETHOTKEY | 0x0 | 0x0 |
| WM_GETICON | 0x0 | 0x0 |
| WM_RENDERFORMAT | 0x0 | 0x0 |
| WM_DRAWCLIPBOARD | 0x0 | 0x0 |
| WM_CHANGECBCHAIN | 0x0 | 0x0 |
| 0x313 | 0x0 | 0x0 |
| WM_THEMECHANGED | 0x0 | 0x0 |
| 0x31b | 0x0 | 0x0 |
| WM_DWMNCRENDERINGCHANGED | 0x0 | 0x0 |
| 0xc05f | 0x0 | 0x0 |
| 0xc060 | 0x0 | 0x0 |

### 4.4.5 Analýza výsledků

Není překvapením, že v případě, že UIPI filtr není aktivní, je na cílovou aplikaci doručena valná většina *window messages*. Díky tomu mohou aplikace na stejném *integrity levelu* vzájemně komunikovat a je tak zajištěna flexibilita a jednoduchost užívání grafického rozhraní Windows. Za předpokladu, že jsou běžící aplikace rozděleny do *integrity levelů* dle jejich skutečné důvěryhodnosti, jak by tomu správně ve Windows mělo být, jsou rizika minimalizována. Jakmile je ale nedůvěryhodná aplikace spuštěna se zvýšeným IL, což může nastat v důsledku špatné konfigurace operačního systému či uživatelské chyby, dochází k ohrožení, neboť získá přístup k celému API poskytovanému *window messages*. Kromě toho se jí také otvírá možnost provádět další operace, které jsou při interakcích napříč různými IL blokovány, jako je např. injekce DLL nebo používání *thread hooks* a *journal hooks*.

Nicméně i v případě správně nakonfigurovaného systému a obezřetného a disciplinovaného uživatele je možné skrze *window messages* napáchat škody nebo minimálně činit přítěž. Pokud uživatel stáhne nedůvěryhodnou aplikaci z internetu a následně ji spustí, je jí v obecném případě přiřazen IL *medium*. Nedůvěryhodný program tak sice nezíská přístup úrovně administrátora nebo možnost číst citlivá data, neboť s takovými daty by aplikace běžící na IL *medium* neměly pracovat, nicméně může, např. prostřednictvím zpráv jako jsou `WM_QUIT` a `WM_DESTROY`, spustit *denial-of-service* útok, kdy rozesíláním těchto zpráv bude ukončovat všechny programy a zavírat všechna okna, která mu to dovolí.

Další poznatky z experimentů jsou na poli samotného fungování *window messages* a UIPI filtru. V sekcích 4.4.4.1 a 4.4.4.2 bylo poukázáno na to, že v určitých případech předchází kontrola prostupnosti UIPI filtru validaci parametrů *window messages*. Tento závěr byl učiněn na základě porovnání experimentů A.5.1.2 a A.5.2.2, u kterých je možné pozorovat rozdíly v chování funkce `PostMessage` při deaktivovaném a aktivovaném UIPI. Byly identifikovány situace, kdy *window messages*, které bez aktivovaného UIPI hlásily chybový kód `0x487`, po aktivaci jednoznačně indikovaly blokaci UIPI skrze kód `0x5`.

U jiných *window messages* však bylo možné pozorovat opačnou tendenci. Mezi ně patří zprávy, které bez UIPI skončily s chybovým kódem `0x57` nebo `0x3ea`. V jejich případě nezískal po aktivaci UIPI chybový kód `0x5` prioritu nad jejich stávajícími chybovými kódy, přestože není UIPI filtr pro tyto zprávy průchozí [35, 82].

Za pozornost stojí také celá řada nedokumentovaných *window messages*, které se pomocí manuálního testování podařilo identifikovat jako dodatečně doručené či dodatečně blokované. To znamená, že tyto zprávy kontrolují obsah svých parametrů, což svědčí o tom, že pravděpodobně mají přiřazen nějaký význam a využití, které buď není veřejné, nebo pro něj chybí dokumentace.

Důležitým bodem, zejména pro potřeby této práce, který z experimentů vyplynul, bylo, že pro testování UIPI filtru *window messages* se jako spolehlivější zdroj informací jeví funkce `PostMessage`. Jak je nastíněno v sekcích 4.4.4.1 a 4.4.4.2, chybové kódy, které indikují chybu v obsahu parametrů *window message*, se při použití `PostMessage` vyskytovaly pouze minimálně. Nevýhodou `PostMessage`však je, že některé *window messages* musí být posílány pomocí funkce `SendMessage` a při použití `PostMessage` skončí jejich odeslání s chybovým kódem `0x487`.

Jedním z pozoruhodných jevů, které se při experimentování projevily, byla situace, kdy jedna *window message* vygenerovala několik dalších. Nejčastěji se jednalo o *window messages* `0x313`, `WM_DWMCOMPOSITIONCHANGED` a `WM_DWMNCRENDERINGCHANGED`, které při svém prvním použití vygenerují celou řadu dalších *window messages*, z nichž některé by samy o sobě nemohly být při aktivovaném UIPI doručeny. Mezi další zprávy, které tento jev dokáží způsobit, patří zprávy spojené s *input method editorem* (IME), jejichž názvy jsou prefixovány `WM_IME`.

### 4.4.6 Analýza doručených *window messages*

Hlavním výstupem experimentů je seznam zpráv v tabulkách 4.1 a 4.2, jejichž doručení je možné i při aktivním UIPI filtru. Kromě zpráv, jež jsou v tomto kontextu běžně uváděny v literatuře [35, 82] se podařilo identifikovat další zprávy, které literatura vynechává. Níže jsou zprávy rozebrány do většího detailu a je diskutován jejich potenciál pro zneužití.

#### 4.4.6.1 `WM_NULL`

Zpráva `WM_NULL` je speciální v tom, že nenese žádnou informaci a výchozí reakcí programů by mělo být ji ignorovat [152]. Své využití nachází zejména při testování, zda cílové okno reaguje na *window messages* [152]. Pokud je aplikace připojená k jiné aplikaci pomocí *hooku*, může změnou konkrétní zprávy na `WM_NULL` pomocí funkcí Windows API zařídit, aby byla daná zpráva ignorována [152]. `WM_NULL` tedy sama o sobě nenabízí prostor pro zneužití. Avšak v případě, že by aplikace implementovala vlastní logiku, která tuto zprávu zpracovává, dalo by se o zneužití spekulovat. Tyto případy jsou však silně závislé na konkrétní posuzované aplikaci.

#### 4.4.6.2 `WM_MOVE` a `WM_SIZE`

*Window messages* `WM_MOVE` resp. `WM_SIZE` jsou oknům posílány poté, co je změněna jejich pozice [153] resp. velikost [154]. V dnešní době se už od jejich využití a využívají se zejména `WM_WINDOWPOSCHANGING` a `WM_WINDOWPOSCHANGED` [155] v kombinaci s dalšími zprávami generovanými funkcí `SetWindowPos`. Podpora pro starší aplikace je nadále udržována tak, že funkce `DefWindowProc` přeloží *window message* `WM_WINDOWPOSCHANGED` právě na dvojici zmiňovaných zpráv `WM_MOVE` a `WM_SIZE`. Novější aplikace však na tyto zprávy nereagují a pro řízení své velikosti používají hlavně `WM_WINDOWPOSCHANGED`, která není UIPI filtrem propouštěna. Pokud tedy aplikace neimplementuje vlastní logiku, která by byla závislá na `WM_MOVE` nebo `WM_SIZE`, nemělo by být možné manipulovat prostřednictvím těchto zpráv s jejím oknem.

#### 4.4.6.3 `WM_GETTEXT` a `WM_GETTEXTLENGTH`

Dvojice zpráv `WM_GETTEXTLENGTH` a `WM_GETTEXT` lze použít pro čtení textu cílového okna. Typicky se nejprve použije funkce `WM_GETTEXTLENGTH`, která zjistí délku daného řetězce [156], následně je alokován paměťový blok této velikosti a poté je pomocí *window message* `WM_GETTEXT` řetězec

do paměťového bloku načten [157]. To, že se tyto zprávy nacházejí v tabulce 4.1 implikuje, že je možné je pomocí `SendMessage` posílat oknům napříč *integrity levely*.

Experiment A.6 tuto skutečnost potvrzuje. V tomto experimentu je učiněn pokus o vyčtení hesla z jednoduchého okna pro zadávání přihlašovacích údajů vytvořeného pomocí PowerShell [158]. Závěrem experimentu je, že zpráva `WM_GETTEXT` neuspěje, neboť políčko, do kterého je heslo zadáváno, je pro tento účel přizpůsobeno[2] a brání svůj obsah proti čtení. Nicméně zpráva `WM_GETTEXTLENGTH` zůstává plně funkční a lze s její pomocí získat délku zadaného hesla.

Tento příklad demonstruje, že i *window messages*, které nedokáží modifikovat data, můžou být být zneužity. Jakmile má útočník k dispozici délku hesla uživatele, může tuto informaci využít pro zúžení prohledávaného prostoru při následujícím *brute-force* útoku. Pokud by navíc vlivem programátorské chyby nebylo políčko pro heslo uzpůsobeno tak, aby chránilo zadávaný obsah, bylo by možné pomocí zprávy `WM_GETTEXT` heslo přímo přečíst.

#### 4.4.6.4 WM_GETHOTKEY a WM_GETICON

`WM_GETHOTKEY` a `WM_GETICON` se, podobně jako předchozí dvojice zpráv, dají využít pro získání informací o cílovém okně. Oproti `WM_GETTEXT` a `WM_GETTEXTLENGTH` se však liší tím, že namísto textu obsaženého v jednotlivých ovládacích prvcích vrací buď klávesovou zkratku spojenou s daným oknem v případě `WM_GETHOTKEY` [162], nebo ikonu daného okna v případě `WM_GETICON` [163]. Potenciál pro zneužití těchto zpráv je tedy minimální až naprosto žádný.

#### 4.4.6.5 WM_RENDERFORMAT, WM_DRAWCLIPBOARD a WM_CHANGECBCHAIN

*Window messages* `WM_RENDERFORMAT`, `WM_DRAWCLIPBOARD` a `WM_CHANGECBCHAIN` se používají pro práci se schránkou. První z nich, `WM_RENDERFORMAT`, je doručena aktuálnímu vlastníkovi schránky v případě, že dříve odložil dodání obsahu schránky, který je nyní požadován [164]. Za vlastníka schránky je považováno okno, které je spojeno s jejím aktuálním obsahem [165]. Okno se jím typicky stává v momentu, kdy do schránky vloží data, a zůstává jím, dokud není zavřeno nebo jiné okno schránku nevyprázdní [165]. Odložení dodání dat dosáhne okno tak, že funkci `SetClipboardData` zavolá s parametrem `hData` nastaveným na hodnotu `NULL` [165]. Tato funkcionalita se hodí zejména v případě, že je okno schopné data poskytnout v několika formátech, přičemž dodání dat v některých z nich je časově náročné [165]. Jakmile okno provede volání funkce `SetClipboardData` tímto způsobem, přihlásí se k příjmu zpráv `WM_RENDERFORMAT` a v reakci na ně je povinno data v daném formátu dodat [164].

Dokumentace specifikuje, že schránka by měla být používána pouze na požadavek uživatele a aplikace by k ní neměly přistupovat bez jeho vědomí [166]. Z toho důvodu nelze očekávat, že by správně naprogramovaná aplikace skrze schránku uvolnila citlivé informace. V obecném případě by měly pomocí `WM_RENDERFORMAT` jít získat pouze data, která by aplikace v každém případě do schránky vložila. Důvodem pro použití tohoto mechanismu je odložit vložení dat do schránky pro případ, že nakonec požadována nebudou a vyhnout se tak časově náročným výpočtům. Pravděpodobně neexistuje spolehlivý způsob, jak tuto zprávu zneužít, nicméně není vyloučeno, že je to možné. Pokud je část aplikace reagující na zprávu `WM_RENDERFORMAT` implementována špatně, může docházet k chybám, např. v situaci, kdy příjmu této *window message* nepředcházelo volání funkce `SetClipboardData`.

Za zmínku stojí, že UIPI filtr není prostupný pro zprávu `WM_RENDERALLFORMATS` s podobnou funkcionalitou. Tato zpráva se použije v případě, že okno, které je vlastníkem schránky a zároveň odložilo dodání dat, je zavřeno [167]. Účelem této zprávy je zajistit dodání dat tak, aby zůstala dále dostupná pro ostatní aplikace využívající schránku [165]. To však neznamená, že okna pracující se schránkou, která běží na vyšších *integrity levelech*, by nebyla kvůli UIPI schopna před svým zavřením dodat do schránky data. Protože původcem této zprávy bude s nejvyšší prav-

[2] Při vytvoření tohoto okna byl použit styl `ES_PASSWORD` [159, 160, 161].

děpodobností samotný operační systém, dojde k jejímu úspěšnému doručení a data budou před zavřením okna do schránky vložena.

Zbývající dvě *window messages*, `WM_DRAWCLIPBOARD` a `WM_CHANGECBCHAIN`, se vztahují k oknům zobrazujícím aktuální obsah schránky. V případě, že takových oken existuje více, jsou vzájemně propojena do řetězce, přičemž každý článek je povinen si uložit *handle* okna představujícího následující článek [166]. Zobrazování obsahu schránky pak funguje tak, že první článek v řetězci obdrží zprávu `WM_DRAWCLIPBOARD`, aktualizuje svůj obsah a následně předá tuto zprávu dalšímu článku [166].

Podobně jako u *window message* `WM_RENDERFORMAT` ani zde pravděpodobně neexistuje způsob, kterým by se dala zpráva `WM_DRAWCLIPBOARD` spolehlivě zneužít. Pokud je však dodání obsahu schránky náročné na výpočet a řetězec oken, která jej zobrazují, velmi dlouhý, šlo by teoreticky `WM_DRAWCLIPBOARD` zneužít pro přetížení oken zodpovědných za dodání obsahu nebo, v extrémních případech, operačního systému.

O něco zajímavější je zpráva `WM_CHANGECBCHAIN`, která se používá, když je okno odstraňováno z řetězce oken zobrazujících obsah schránky [166]. V takovém případě je operačním systémem `WM_CHANGECBCHAIN` doručena prvnímu oknu v řetězci a podobně jako `WM_RENDERFORMAT` je předávána mezi jednotlivými články, dokud se nedostane k oknu, které předchází odstraňované okno [166]. V ten moment má okno zpracovávající zprávu `WM_CHANGECBCHAIN` povinnost následující okno z řetězce vyřadit [168] a na jeho místo zařadit okno, jehož *handle* se nachází v `lParam` [169].

Útočník by tedy mohl pomocí *window message* `WM_CHANGECBCHAIN` odstraňovat či přidávat okna zobrazující obsah schránky. Nicméně obě funkcionality už pokrývají Windows API funkce `SetClipboardViewer` a `ChangeClipboardChain` a pravděpodobně se tedy jedná o vědomé rozhodnutí učiněné při návrhu API. Útočník má tedy tímto způsobem možnost vyřadit z provozu aplikace sloužící pro zobrazování obsahu schránky nebo v řetězci vytvořit smyčky a tím aplikace přetížit.

#### 4.4.6.6 `0x313` a `0x31b`

První nedokumentovanou zprávou, která dokáže projít filtrem *window messages*, je zpráva `0x313`. Přestože pro ní v oficiální dokumentaci neexistuje stránka, některé internetové zdroje ji spojují se jménem `WM_SYSMENU` [170]. V experimentech A.5.1.1 a A.5.2.1, tedy experimentech používajících `SendMessage`, bylo pozorováno, že `0x313` způsobí doručení několika dalších *window messages*, což svědčí o tom, že má ve Windows své využití. Lze si také povšimnout, že je doručena pokaždé, když je za držení klávesy *shift* kliknuto pravým tlačítkem myši na ikonku programu na hlavní liště Windows GUI, a to včetně všech zpráv, jejichž doručení způsobuje. Protože není známa její aplikace ani významy jejích parametrů, nelze jednoznačně určit, zda by ji bylo možné zneužít.

Druhá zpráva neznámého významu, kterou UIPI filtr propustí, je zpráva `0x31b`. O ní je nejsou dostupné téměř žádné informace s výjimkou její možné asociace se jménem `WM_UAHINIT` [170]. Stejně jako u `0x313` ani u `0x31b` není možné rozhodnout o její zneužitelnosti.

#### 4.4.6.7 `WM_THEMECHANGED` a `WM_DWMNCRENDERINGCHANGED`

*Window messages* `WM_THEMECHANGED` a `WM_DWMNCRENDERINGCHANGED` nevyžadují po oknu či aplikaci, která je přijímá, žádná data. Zpráva `WM_THEMECHANGED` je doručena všem oknům poté, co dojde ke změně motivu uživatelského rozhraní [171]. Motivem se v tomto kontextu rozumí vizuální nastavení GUI, jako jsou barvy, průhlednost, tapeta a podobně. Dle dokumentace je od okna očekáváno, že voláním funkcí `CloseThemeData` a `OpenThemeData` znovu načte informace o motivu a patřičně upraví své vykreslování [171]. Ani u této zprávy pravděpodobně neexistuje způsob, jak ji spolehlivě zneužít. Z dokumentace není zřejmé, jak výpočetně náročné je volání funkcí `CloseThemeData` a `OpenThemeData` nebo následující úprava vykreslování, a není tak možné rozhodnout, zda by `WM_THEMECHANGED` a `WM_DWMNCRENDERINGCHANGED` mohly umožňovat DoS. Jako

u většiny ostatních doručených *window messages* by i u těchto zpráv bylo teoretické zneužití specifické pro konkrétní aplikaci a závislé na implementaci logiky spojené s jejich zpracováním.

Zpráva `WM_DWMNCRENDERINGCHANGED` spadá do skupiny zpráv spojených s *Desktop Window Managerem* (DWM) [172], části grafického subsystému Windows, která řídí vykreslování na ploše [173]. DWM bylo zavedeno společně s MIC a UIPI ve Windows Vista a poskytuje funkcionality jako průhledné okraje oken, animace a podporu pro vysoké rozlišení [173]. *Window message* `WM_DWMNCRENDERINGCHANGED` je oknům doručena poté, co dojde ke změně zásad spojených s vykreslováním částí oken, jež spravuje DWM [172]. Mezi ně patří právě rámečky, ikona nebo tlačítka pro zavření a minimalizaci okna [174]. Dokumentace už nespecifikuje, jestli a jak by okno mělo na přijetí této zprávy reagovat, a není ani jasné, jak se při jejím přijetí chová `DefWindowProc`. Zneužití `WM_DWMNCRENDERINGCHANGED` by tak opět pravděpodobně bylo vázáno na případnou vlastní logiku, která tuto zprávu zpracovává.

#### 4.4.6.8 `0xc0a1`, `0xc0a2`, `0xc05f` a `0xc060`

Zprávy `0xc0a1`, `0xc0a2`, `0xc05f` a `0xc060` spadají podle tabulky 1.2 do kategorie řetězcových zpráv, které, jak je rozebíráno v sekci 1.3, lze registrovat pomocí funkce `RegisterWindowMessage`. Ta je volána s řetězcem, který registrovanou zprávu bude popisovat, a v návratové hodnotě pak volající obdrží číselný identifikátor [175] pro použití ve funkcích `SendMessage` a `PostMessage`.

Pomocí nástroje Spy++ [139] je možné řetězce použité při registraci zpětně vyčíst. Tímto způsobem lze zjistit, že nižší z dvojice zpráv koresponduje s řetězcem `MSUIM.Msg.LangBarModal` a vyšší s řetězcem `MSUIM.Msg.Private`. Je tedy zřejmé, že zprávy `0xc0a1` a `0xc05f` resp. `0xc0a2` a `0xc060` jsou ekvivalentní. Pozorované identifikátory se v experimentech A.5.2.1 a A.5.2.2 lišily pravděpodobně z toho důvodu, že oba experimenty probíhaly v jiné relaci operačního systému, tedy došlo mezi nimi k restartování počítače. V důsledku toho jim funkce `RegisterWindowMessage` pokaždé přiřadila jiné identifikátory.

Podle některých internetových článků [176, 177] pocházejí `MSUIM` zprávy z knihovny `MSCTF.dll`, které společně s `ctfmon.exe` tvoří součást *Text Services Frameworku* (TSF), což je Windows služba spravující zadávací metody, rozložení klávesnice, zpracování textu a poskytující podporu pro znakově komplikované jazyky jako např. čínština nebo japonština [177, 175].

Pokaždé, když je vytvořeno nové okno, zavolá operační systém funkci, která provede automatické načtení CTF klienta, jehož součástí je i registrace analyzovaných *window messages* [177]. Klient se následně připojí na CTF server realizovaný v procesu `ctfmon.exe` [177]. Komunikace pak probíhá tak, že server kontinuálně přijímá zprávy od klientských aplikací, ale aplikace začnou příchozí komunikaci zpracovávat až tehdy, když jsou upozorněny pomocí registrovaných *window messages* [177].

Nicméně hlavním předmětem článku [177] je zranitelnost nalezená v protokolu používaném CTF. Tato zranitelnost, označená CVE-2019-1162 [178], zneužívala toho, že CTF nezahrnuje žádnou formu autentizace a je tak možné podvrhnout CTF server a donutit privilegované aplikace navázat spojení. Protože se CTF protokol datuje až k operačnímu systému Windows XP a od té doby byl ve všech verzích Windows používán téměř beze změny, podařilo se autorovi článku, Tavisovi Ormandymu, identifikovat způsob, kterým je možné spustit libovolný program s oprávněními cílové aplikace.

Společně s článkem zveřejnil Ormandy ještě interaktivní aplikaci `ctftool` [179], s jejíž pomocí je útok možné vykonat. Součástí aplikace je skript, který požádá operační systém o zvýšená oprávnění, čímž je vytvořeno UAC dialogové okno běžící pod uživatelem `NT AUTHORITY\SYSTEM`, a následně pod tímto uživatelem pomocí zranitelností CTF protokolu spustí privilegovaný *shell* [177]. Útok se podařilo reprodukovat na operačním systému Windows 10 verze 1903, *build* 18362.30. V současné verzi Windows 11 ho už ale není možné provést, neboť Microsoft zranitelnost v roce 2019 opravil [180].

# Kapitola 5
# Doporučení

Přestože se může jevit, že *Shatter* a jeho varianty jsou dávno vyřešené hrozby, které byly odstíněny technologiemi *User Interface Privilege Isolation* a *Mandatory Integrity Control*, stále existují rizika, která mohou ohrozit bezpečnost aplikací či uživatelských dat. Nejedná se ovšem o rizika s tak drastickými dopady, jako měla původní podoba útoků typu *Shatter*, jejichž prostřednictvím bylo možné dosáhnout *privilege escalation*, ale spíše o rizika plynoucí z nesprávné konfigurace operačního systému nebo špatného zacházení se zavedenými ochranami.

Objevení zranitelnosti, kterou *Shatter* zneužívá, byla pravděpodobně pouze otázka času. Systém předávání *window messages* byl zaveden v dobách, kdy při vývoji operačních systémů nebyl kladen přílišný důraz na bezpečnost. Nicméně to, že i po zavedení UIPI byly *window messages* zneužity pro tak závažný útok, jako je útok na CTF popisovaný v sekci 4.4.6.8, vytváří prostor pro spekulaci, zda v budoucnu nebudou objeveny další zranitelnosti systému předávání *window messages*. Z toho důvodu je důležité, aby jak uživatelé, tak vývojáři aplikací byli obeznámeni s riziky spojenými s tímto systémem a dokázali se jim vyhnout.

## 5.1 Přihlášení k ochranám

Zásadní výhodou ochran diskutovaných v této práci, *Mandatory Integrity Control* a *User Interface Privilege Isolation*, je, že ve své základní formě jsou aktivní bez jakéhokoliv přičinění uživatele. Na rozdíl od bezpečnostních opatření, jako jsou např. *Address Space Layout Randomization* (ASLR) nebo *Data Execution Prevention* (DEP), neexistuje ve Windows žádný ovládací panel, na kterém by se MIC a UIPI dalo zapínat či vypínat.

Jediná část UIPI, která je konfigurovatelná, je řízení *UIAccess*. V sekci 4.2 je uvedeno, že pomocí *Local Security Policy* lze konfigurovat, zda se aplikace žádající o atribut *UIAccess*, musí nacházet v jedné ze složek, do kterých může zapisovat pouze administrátor. Toto nastavení lze nalézt ve stromě `Local Policies\Security Options` pod jménem „*Only elevate UIAccess applications that are installed in secure locations,*“ a je doporučeno jej ponechat na výchozím nastavení, kterým je hodnota *Enabled*. Dále se zde nachází nastavení „*Allow UIAccess applications to prompt for elevation without using the secure desktop,*“ které by *UIAccess* aplikacím umožnilo žádat o zvýšení oprávnění bez použití *secure desktop*. Toto nastavení není vhodné zapínat a jeho hodnota by tedy měla setrvat na volbě *Disabled*. *Secure desktop* je speciální tím, že na ní mohou běžet pouze důvěryhodné programy spuštěné pod uživatelem `NT AUTHORITY\SYSTEM` [181], které jsou izolovány od běžných uživatelských aplikaci. Alternativou je, že by *UIAccess* aplikace žádaly o zvýšení oprávnění prostřednictvím *interactive desktop*, což, jak již bylo řečeno v sekci 2.3, s sebou historicky neslo značná rizika.

Při stažení nedůvěryhodného souboru z internetu a jeho spuštění je vznikajícímu procesu přiřazen výchozí IL *medium*. Samo o sobě se nejedná o zranitelnost nebo situaci, která by přinášela závažná bezpečnostní rizika, ale pro optimalizaci zabezpečení a maximalizaci ochrany by bylo lepší danou aplikaci spustit s IL menším než *medium*, jako např. *low* nebo *untrusted*. Případně je možné na nižší IL nastavit celou složku se staženými soubory a do hlavičky ACE vyjadřující IL přidat příznaky, které zajistí, že IL bude děděn všemi podobjekty v této složce.

Jak je naznačeno v sekci 4.4.5, i aplikace běžící s IL *medium* mohou v operačním systému napáchat škody. Příkladem je situace, kdy má uživatel na svých datech příliš volně nastavené DACL. Pokud takové soubory nemají vyšší *integrity level* než *medium*, mohou být zpřístupněny útočníkem. Druhým scénářem zneužití je případ, kdy stažená nedůvěryhodná aplikace začne hromadně rozesílat zprávy `WM_QUIT` a `WM_DESTROY`, s jejichž pomocí dokáže zavřít okna a ukončit procesy běžící na stejném IL a dosáhnout tak DoS útoku.

Důležité je však poznamenat, že, jak demonstruje experiment A.3, ne všechny aplikace musí být kompatibilní s *integrity levely low* a *untrusted*. Je očekávatelné, že řada aplikací nebude schopna s těmito *integrity levely* běžet nebo poběží s omezenými funkcionalitami. Požadavek na běh při nízkém *integrity levelu* musí být většinou začleněn již do fáze návrhu aplikace, aby došlo k jejímu přizpůsobení.

Optimalizaci konfigurace může uživatel provést i na úrovni souborů. V sekci 3.2.6, která rozebírá chování *integrity levelů* v souborovém systému, bylo zjištěno, že výchozí nastavení *policies* není dostačující pro ochránění souboru před přesunutím nebo smazáním. Podobně jako předchozí diskutovaný scénář, je i zde zneužití podmíněno špatným nastavením přístupových oprávnění. Za běžných okolností jsou při vytváření souborů nastavována DACL automaticky operačním systémem a zneužití, jak této situace, tak těch popisovaných výše, by nebylo možné. Nelze jej však vyloučit, neboť není vzácností, že se uživatelé při konfiguraci dopouští chyb.

Velmi důležité je také vybrat správný souborový systém. Při automatické instalaci Windows nebo zakoupení stroje s předinstalovaným systémem je uživatel tohoto rozhodnutí zproštěn, neboť ve všech těchto případech bude Windows používat NTFS, který podporuje *security descriptory*. Jak již bylo řečeno v sekci 3.2.6, *security descriptory* jsou naprosto nezbytné pro fungování *integrity levelů* a všech bezpečnostních prvků na nich závislých. Pro optimální zabezpečení je tedy radno nevyužívat souborové systémy, které nepodporují IL, jako např. souborové systémy z rodiny FAT. U nich bylo navíc v experimentech A.1 identifikováno chování, při němž není aplikována výchozí *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP`, která omezuje přístup subjektů s nižší integritou k *securable objectům* s vyšší integritou.

Posledním bodem je častá aktualizace operačního systému, zejména jeho bezpečnostních záplat. Jak bylo řečeno v úvodu této kapitoly, nelze vyloučit, že v budoucnu dojde k zneužití *window messages* pro novou formu útoku. Nejvíce rizikové je však používání verzí Windows, které již nejsou podporovány, jako např. Windows 1903 se zranitelností CTF protokolu zmiňovanou v sekci 4.4.6.8.

## 5.2 Doporučení pro vývojáře aplikací

Při práci s *window messages* je nejdůležitější si uvědomit, že neexistuje způsob, jakým by konkrétní zpráva mohla být přiřazena ke konkrétnímu odesílateli. Jinými slovy, mechanismus předávání *window messages* neobsahuje žádnou formu autentizace. Proto by k přijímaným *window messages* mělo být přistupováno jako ke každému jinému uživatelskému vstupu a pokud je to možné, prováděna jejich důkladná validace.

Toto doporučení nelze zcela aplikovat na typy *window messages* jako takové. U příchozí *window message* se sice může aplikace rozhodnout, zda ji zpracuje či nikoliv, ale v případě, že se rozhodne pro druhou volbu, např. u zprávy `WM_MOUSEMOVE`, která nese informaci o pohybech kurzoru [182], může tím výrazně omezit svou použitelnost. Aplikace tohoto doporučení je však značně důležitá u parametrů *window messages*. Z provedených experimentů je patrné, že s pomocí programu `sender.exe` lze libovolnému oknu odeslat libovolnou zprávu s libovolnými parametry.

Pokud tedy aplikace spoléhá na určitý formát dat obsažených v parametru, může dojít k tomu, že její předpoklady nebudou naplněny. Tímto se aplikace otevírá běžným rizikům, které plynou z nedostatečné validace nedůvěryhodného uživatelského vstupu, jako jsou útoky typu *injection*.

V případě, že se vývojář rozhodne jako formu ochrany použít přímou interakci s UIPI filtrem a blokaci některých zpráv, měl by upřednostnit funkci `ChangeWindowMessageFilterEx`. Jak bylo rozebráno v sekci 4.3, její předchůdce, funkce `ChangeWindowMessageFilter`, aplikuje úpravy filtru na celý proces namísto konkrétního okna [128], což nemusí být vždy žádoucí. Navíc se kvůli neintuitivnímu pojmenování *flagů*, které se používají pro indikaci, zda má zpráva být filtrována či propouštěna [128], vývojář vystavuje zbytečnému riziku lidské chyby. Dalším úskalím v používání této funkce může být situace, kdy se vývojář pokusí zablokovat zprávu ze skupiny zpráv, které UIPI filtrem projdou vždy bez ohledu na jeho nastavení. V tomto případě totiž `ChangeWindowMessageFilter` vrací stejnou hodnotu jako při úspěchu [128], což může vést ke zmatení. Funkce `ChangeWindowMessageFilterEx` obsahuje revidované rozhraní s přejmenovanými *flagy* a navíc dokáže poskytnout rozšířené informace o výsledku jejího volání [129].

Stejně jako u předchozího doporučení se i zde může stát, že odfiltrováním některých *window messages* aplikace negativně ovlivní svou použitelnost. Opačný případ, tedy případ, kdy vývojář umožní některým *window messages* procházet UIPI filtrem, také není bez rizika. Pamatovat je nutné zejména na to, že množina zpráv, pro které je filtr konkrétní aplikace průchozí, je téměř veřejná informace. V rámci minut lze, např. pomocí `sender.exe`, vyzkoušet na cílovou aplikaci poslat všech 65 536 *window messages* a pomocí funkce `GetLastError` zjistit, které prošly a které nikoliv.

Zásadní je také dbát zvýšené opatrnosti při implementaci vlastní logiky zpracovávající konkrétní typ *window messages*. U velké části zpráv propuštěných UIPI filtrem, které jsou analyzovány v sekci 4.4.6, nebyl přímo identifikován způsob jejich zneužití, nicméně úprava mechanismu jejich zpracování s sebou přináší riziko programátorských chyb, které by aplikaci mohly učinit zranitelnou. Pokud se vývojář k tomuto řešení uchýlil s původním záměrem komunikace s ostatními okny či procesy, měl by namísto *window messages* použít jiné formy IPC, jako např. *remote procedure calls*, *named pipes* nebo *Component Object Model* [57].

Závěrem je nutné upozornit na *window messages* `WM_GETTEXT` a `WM_GETTEXTLENGTH`. Tyto zprávy patří do skupiny zpráv, které jsou za běžných okolností propouštěny UIPI filtrem i napříč *integrity levely*. Jak demonstruje experiment A.6, je s jejich pomocí skutečně možné číst data cizího okna. V tomto experimentu byly zprávy použity v pokusu vyčíst uživatelské jméno a heslo z dialogového okna. Zpráva `WM_GETTEXT` narazila na to, že políčko obsahující heslo bylo pro své využití uzpůsobeno a neuvolnilo svůj obsah. Nicméně pomocí zprávy `WM_GETTEXTLENGTH` bylo možné vyčíst délku hesla. Vývojáři by tedy měli mít na paměti, že i zprávy tohoto rázu patří mezi zprávy, pro které je filtr UIPI průchozí. Pokud v některém z dialogových oken svých programů pracují s citlivými daty, měli by příslušné ovládací prvky přizpůsobit tak, aby nemohlo dojít k vyčtení jejich obsahu.

# Závěr

Cílem práce bylo zdokumentovat prapůvodní podobu útoku *Shatter*, provést analýzu jeho možných variant a objasnit principy, na kterých stojí. Stejným způsobem se práce měla věnovat i bezpečnostním opatřením, která byla v reakci na útoky typu *Shatter* zavedena, a technologiím, jež umožňují jejich fungování. Úkolem praktické části pak bylo navrhnout testovací aplikace, které se využijí při demonstraci toho, jakým způsobem se obrané mechanismy chovají při různých úrovních *integrity levelů*. Jejím hlavním účelem bylo shrnutí poznatků a formulování doporučení pro vývojáře aplikací včetně pokynů, jak se k ochraně přihlásit.

V úvodu práce byly představeny technologie, na nichž útok *Shatter* stojí, a krok po kroku byl demonstrován průběh útoku. V textu práce byl dále věnován prostor osvětlení principů *Mandatory Integrity Control* a *User Interface Privilege Isolation*. V kapitole věnující se MIC bylo identifikováno překvapivé chování *policies*, které při špatném nastavení prostředí umožní neoprávněným subjektům mazat či přesouvat soubory. V rámci stejné kapitoly vznikla i dvojice programů, které uživateli umožňují nastavit na libovolném souboru či procesu kýžený *integrity level* v rozsahu *untrusted* až *system*.

Aplikace nalezly své využití hned v následující kapitole, jejíž součástí byla rozsáhlá analýza filtru *window messages* zavedeného UIPI. Pomocí programů implementujících funkcionalitu nutnou pro přijímání a odesílání *window messages* bylo testováno, pro jaké *window messages* je filtr průchozí, a zároveň srovnáno chování aplikací s aktivovanými a deaktivovanými ochranami. Výsledkem experimentů bylo odhalení tří *window messages*, které byly schopny projít UIPI filtrem, aniž by byla tato skutečnost zdokumentována v literatuře. Součástí této kapitoly byla také hlubší analýza propuštěných *window messages*, ve které se podařilo nastínit scénář zneužití zprávy `WM_GETTEXTLENGTH` pro získání délky hesla zadaného do ovládacího prvku cizího okna.

Závěrečná kapitola využila výsledků experimentů a nově získaných informací pro formulaci doporučení jak pro vývojáře aplikací, tak pro uživatele operačního systému Windows. Dále byl v této kapitole vytyčen způsob, jakým se přihlásit k zavedeným ochranám a nastavit je tak, aby poskytovaly maximální úroveň zabezpečení.

Přestože jsou cíle práce naplněny, existuje prostor pro další rozvíjení tohoto tématu. Nabízí se hned několik oblastí, ve kterých je možné ve výzkumu pokračovat. Mezi ně patří např. téměř nepopsané *integrity levely* jako *protected process* a *secure process*, které byly krátce diskutovány v rešeršní části této práce. Případně by bylo možné doplnit funkcionality aplikace `sender.exe`, např. o možnost automatizovaného testování parametrů *window mesages*.

# Příloha A

# Experimenty

Tato kapitola obsahuje detailní popis průběhů jednotlivých experimentů, jejichž výsledky jsou v práci používány. Zaměřuje se zejména na definici jejich cílů, podrobný popis postupu a použité technologie. Účelem této kapitoly je usnadnit případnou reprodukci experimentů a ověření jejich výsledků.

**Tabulka A.1** Seznam použitého software

| Software | Verze | Popis |
|---|---|---|
| Windows 11 Pro [183] | 23H3, *build* 22631.3296 | operační systém |
| Microsoft C/C++ Optimizing Compiler [113] | 19.38.33134 | C/C++ kompilátor |
| Microsoft Incremental Linker [184] | 19.38.33134 | *linker* |
| Python [114] | 3.12.0:0fb18b0 | Python *runtime* |
| Change mandatory label (`chml`) [108] | 1.53 | nástroj pro čtení a úpravu IL souborů a adresářů |
| Process Explorer [185] | 17.05 | nástroj pro čtení IL procesů |
| PSExec (`psexec`) [112] | 2.43 | nástroj spouštění procesů pod uživatelem `NT AUTHORITY\SYSTEM` |
| Spy++ [139] | 14.00.25420 | nástroj monitorování *window messages* a další interakce s okny programů |
| GFlags [186] | 10.0.22621.0 | nástroj pro řízení nastavení *debugování* a diagnostiky |
| WinDbg [51] | 1.2402.24001.0 | *debugger* pro operační systém Windows |
| Windows PowerShell [158] | 5.1.22621.2506 | PowerShell *runtime* |

## A.1 Chování *integrity levelů* v souborovém systému

Cílem tohoto experiment je pozorovat chování *integrity levelů* v různých souborových systémech. Snaží se zjistit, zda souborové systémy *integrity levely* podporují a jakým způsobem jsou měněny při úpravě, nahrazení, přesunutí a dalších operacích se soubory.

### A.1.1 *New Technology File System*

Následující skupina experimentů se soustředí na *New Technology File System*. Jedná se o nejčastěji využívaný souborový systém ve Windows s vestavěnou podporou pro *security descriptory*, šifrování, kvóty a metadata [187].

#### A.1.1.1 Kompilované programy

Experiment probíhá následovně:

- Vytvoříme demonstrační program `program.cpp` s libovolným obsahem.
- Program zkompilujeme pomocí Microsoft C/C++ kompilátoru.
- Pomocí `chml` nastavíme na výsledném souboru `program.exe` *integrity level low*.
- Pozměníme obsah `program.cpp` a kompilaci opakujeme.

Můžeme pozorovat, že *integrity level* na souboru `program.exe` zůstal zachován. Chování je totožné při použití jiných *integrity levelů*, ať už těch, které používají standardní pojmenované SID, tak u těch, které ne.

#### A.1.1.2 Interpretované programy

Experiment probíhá následovně:

- Vytvoříme demonstrační program `program.py` s libovolným obsahem.
- Pomocí `chml` nastavíme na souboru `program.py` *integrity level low*.
- Pozměníme obsah `program.py`.

Můžeme pozorovat, že *integrity level* na souboru `program.exe` zůstal zachován. Jako v předchozím experimentu je chování totožné při použití jiných *integrity levelů*, ať už těch, které používají standardní pojmenované SID, tak u těch, které ne.

#### A.1.1.3 Operace v souborovém systému

Experiment probíhá následovně:

- Vytvoříme demonstrační soubor `file.txt` s libovolným obsahem.
- Pomocí `chml` nastavíme na souboru `file.txt` *integrity level high*.
- Soubor zkopírujeme, přesuneme či smažeme.

Můžeme pozorovat, že při kopírování souboru není *integrity level* zachován. Dochází totiž k vytvoření nového souboru a aplikaci principů popsaných v sekci 3.2.3. Za předpokladu, že jsme kopírování prováděli z účtu s běžnými uživatelskými oprávněními, bude výsledná kopie bez *integrity levelu*.

Při přesouvání souboru v rámci stejného svazku toto tvrzení už neplatí. V tomto případě je *integrity level* společně s dalšími informacemi jako např. *policies* zachován. Avšak v případě, že je soubor přesouván mezi různými svazky, je hodnota IL i s *policies* zahozena. Překvapivé může být, že soubor s vyšší integritou lze přesouvat z procesu s nižší integritou. Zde jsou na vině *policies*. Výchozí *policy* `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP` není dostačující k tomu, aby omezila

přesouvání souboru. K blokování přesouvání souboru dochází až v tehdy, když je nastavena kombinace `SYSTEM_MANDATORY_LABEL_NO_READ_UP` a `SYSTEM_MANDATORY_LABEL_NO_EXECUTE_UP` *policies*.

U mazání souboru není nutné rozebírat výsledný stav *integrity levelu* a přidružených informací. Nicméně podobně jako výše neubrání výchozí `SYSTEM_MANDATORY_LABEL_NO_WRITE_UP` *policy* soubor s vyšší integritou před smazáním procesem či *trustee* s nižší integritou. Obrany je dosaženo až v situaci, kdy je nastavena *policy* `SYSTEM_MANDATORY_LABEL_NO_READ_UP`.

Chování je totožné při použití jiných *integrity levelů*, ať už těch, které používají standardní pojmenované SID, tak u těch, které ne.

### A.1.2 *File Allocation Table* a *Extensible File Allocation Table*

*File Allocation Table* je starší souborový systém, který byl původně používán na MS-DOS a disketách [188]. Existuje několik typů FAT dle velikosti záznamů v *file allocation* tabulce, řídící struktuře, kterou FAT používá [188], přičemž tento experiment využívá FAT32. Souborový systém exFAT je potom evoluce FAT vyvinutá Microsoftem s podporou pro soubory a disky větších objemů [189]. Žádný z typů FAT ani exFAT nepodporuje *security descriptory* [115].

#### A.1.2.1 Podpora *security descriptorů*

Experiment probíhá následovně:

- Zkopírujeme program `cmd.exe`, který využijeme pro demonstraci.
- Přečteme IL kopie `cmd.exe` pomocí `chml`. Pozorujeme, že `chml` hlásí, že soubor žádný IL nemá a operační systém jej tak interpretuje, jako by měl IL *medium*.
- Na `cmd.exe` nastavíme IL *high*, či jiný libovolný IL (výsledek je totožný).
- Znovu přečteme IL souboru `cmd.exe` pomocí `chml`. Pozorujeme, že soubor stále žádný IL nemá.
- Spustíme soubor `cmd.exe` a sledujeme jeho IL pomocí *Process Exploreru*. Pozorujeme, že běží s IL *medium*.

Můžeme pozorovat, že FAT32 nepodporuje *integrity levely*. Důvod byl stanoven již v úvodu tohoto experimentu, totiž FAT32 neposkytuje podporu pro *security descriptory*, ve kterých se IL *securable objectů* ukládá. Windows tedy pro každý soubor na FAT32 aplikuje výchozí nastavení a interpretuje jej, jako by měl IL *medium*. Naprosto totožným způsobem můžeme postupovat při testování souborového sytému exFAT a dojdeme ke stejným závěrům.

#### A.1.2.2 Výchozí *policies*

Experiment probíhá následovně:

- Vytvoříme demonstrační soubor `file.txt`.
- Spustíme `cmd.exe` s IL *low*.
- Pokusíme se soubor `file.txt` upravit např. příkazem `echo Hello! > file.txt`.

Můžeme pozorovat, že úprava souboru proběhne úspěšně. Závěrem tedy je, že přístup procesu s nižší integritou k *securable objectu* s vyšší integritou není na FAT32 omezen. Stejně jako u předchozího experimentu je i zde průběh a výsledek totožný pro exFAT.

## A.2 Chování *integrity levelů* s funkcí *Run as administrator* a programem psexec

Cílem tohoto experimentu je demonstrovat omezení v nastavování konkrétních *integrity levelů* v rozsazích `0x2001` až `0x2fff` a `0x3001` až `0x3fff`. Experimenty se snaží poukázat na to, že stávající nástroje jako funkcionalita *Run as administrator* a program `psexec` nejsou dostačující, protože nepokrývají uvedená rozpětí IL.

### A.2.1 Funkce *Run as administrator*

Experiment probíhá následovně:

- Zkopírujeme program `cmd.exe`, který využijeme pro demonstraci.
- Na kopii `cmd.exe` nastavíme IL *medium plus* pomocí `chml`.
- Pozorujeme, že při spouštění z účtu se standardními oprávněními je v souladu s předpoklady kvůli *policy* `TOKEN_MANDATORY_NEW_PROCESS_MIN` vznikajícímu procesu přiřazen IL *medium*.
- Spustíme `cmd.exe` skrze funkci *Run as administrator*.

Můžeme pozorovat, že na rozdíl od případu, kdy jsme program spustili z běžného uživatelského účtu, nebylo vznikajícímu procesu přiřazeno minimum z IL nastaveného na souboru a IL spouštějícího procesu, tedy na místo IL *medium plus* získal proces IL *high*. Aplikuje se totiž jeden z principů rozebíraných v sekci 3.2.3 o přiřazování IL, podle kterého získají potomci procesu běžícího s IL *high* také IL *high*.

Pro případ, že potřebujeme proces spustit s IL *high*, je sice funkce *Run as administrator* vhodná, nicméně v případě, že je nutné procesu přiřadit některý z nestandardních IL v rozmezí `0x2001` až `0x2fff` či IL *medium plus*, nejsou její funkcionality dostačující.

### A.2.2 Program psexec

Experiment probíhá následovně:

- Zkopírujeme program `cmd.exe`, který využijeme pro demonstraci.
- Na kopii `cmd.exe` nastavíme IL `0x3500` pomocí `chml`.
- Opět pozorujeme, že při spuštění z účtu se standardními oprávněními je procesu přiřazen IL *medium*.
- Pomocí `psexec` spustíme `cmd.exe` pod účtem `NT AUTHORITY\SYSTEM`.

Můžeme pozorovat, že podobně jako v experimentu A.2.1 je vznikajícímu procesu přiřazen IL *system* a nikoliv nastavený minimální IL `0x3500`. Pro případy, že proces potřebujeme spustit s nestandardními IL v rozmezí `0x3001` až `0x3fff`, není `psexec` vhodnou volbou.

## A.3 Spouštění procesů s *integrity levely untrusted* a *low*

Cílem tohoto experimentu je porovnat průběh spouštění programů s IL *medium* a *untrusted* či *low*. Experiment se dále snaží poukázat na problémy, které při spouštění programů s IL nižšími než *medium* mohou nastat a diagnostikovat jejich příčinu. Pro dosažení cílového IL na spouštěných procesech se experiment opírá o *policy* `TOKEN_MANDATORY_NEW_PROCESS_MIN`, která je rozebírána v sekcích 3.2.1.1 a 3.2.5.2.

### A.3.1 *Untrusted* a *low integrity levely*

Experiment probíhá následovně:

- Zkopírujeme program `calc.exe`, který využijeme pro demonstraci.
- Pomocí `chml` nastavíme na kopii `calc.exe` IL *untrusted* případně *low* (pro potřeby tohoto experimentu je výsledek stejný).
- Pomocí programu GFlags nastavíme na procesu příznak *show loader snaps*, který zapříčiní, že při *debugování* programu uvidíme výstupy Windows DLL *loaderu* [190].
- Spustíme `calc.exe` v programu WinDbg.

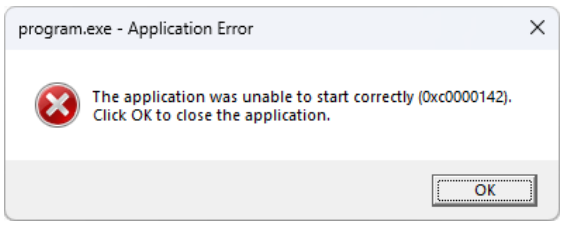
Můžeme pozorovat, že při krokování programu ve WinDbg se postupně objeví několik chybových hlášek, ze kterých se dozvídáme, že se nezdařilo provést inicializační rutinu při připojování knihovny `User32.dll`. Závěrem je, že inicializace celého procesu selhává s chybovým kódem `0xc0000142`, který je později zobrazen i v dialogovém okně na obrázku A.1. Nejdůležitější z chybových výpisů jsou zobrazeny ve výpisu kódu A.1. Průběh celého experimentu je totožný i v případě, že na `calc.exe` nastavíme IL *low* a výsledkem tedy je, že s těmito IL `calc.exe` nedokáže běžet.

Ve WinDbg lze pomocí příkazu `!error` zjistit význam chybového kódu `0xc0000142`. S ním je spjata chybová zpráva „*Initialization of the dynamic link library %hs failed. The process is terminating abnormally*“ [51], která naznačuje, že *linkování* dynamické knihovny, konkrétně `user32.dll`, selhalo a proces bude ukončen.

**Výpis kódu A.1** Výpisy WinDbg při spouštění procesu s IL *untrusted* [51]

```
0c9c:2714 @ 14159984 - LdrpInitializeNode - ERROR: Init routine 00007FFF5432B2A0
for DLL "C:\Windows\System32\USER32.dll" failed during DLL_PROCESS_ATTACH
(...)
0c9c:2714 @ 14214968 - LdrpInitializeProcess - ERROR: Running the init routines
of the executable's static imports failed with status 0xc0000142
(...)
0c9c:2714 @ 14245078 - _LdrpInitialize - ERROR: Process initialization failed
with status 0xc0000142
```

**Obrázek A.1** Chyba při spouštění procesu s *integrity levelem untrusted* [183]




### A.3.2 *Medium* a vyšší *integrity levely*

Experiment probíhá následovně:

- Zkopírujeme program `calc.exe`, který využijeme pro demonstraci.

- Pomocí programu GFlags nastavíme na procesu příznak *show loader snaps*.
- Spustíme `calc.exe` v programu WinDbg.

Můžeme pozorovat, že inicializace procesu proběhne v pořádku a knihovna `User32.dll` je úspěšně načtena. To potvrzují i zprávy Windows DLL *loaderu* zobrazené ve WinDbg ve výpisu kódu A.2, jež ukazují úspěšné načtení funkcí `MonitorFromWindow` a `MonitorFromPoint`, které jsou implementovány právě v `User32.dll` [191, 192]. Proces se úspěšně spustí a je zobrazeno okno kalkulačky.

**Výpis kódu A.2** Výpisy WinDbg při spouštění procesu s IL *medium* [51]

```
0694:53b8 @ 15804531 - LdrpLoadDllInternal - ENTER: DLL name: C:
\Windows\SYSTEM32\user32.dll
0694:53b8 @ 15804531 - LdrpLoadDllInternal - RETURN: Status: 0x00000000
0694:53b8 @ 15804531 - LdrpGetProcedureAddress - INFO: Locating procedure
"MonitorFromWindow" by name
0694:53b8 @ 15804531 - LdrpGetProcedureAddress - INFO: Locating procedure
"MonitorFromPoint" by name
```

## A.4 Kontrola platnosti *handle* okna s vyšším *integrity levelem*

Cílem tohoto experimentu je demonstrovat, že omezení kontroly platnosti *handle* okna s vyšším *integrity levelem*, které je vymáháno UIPI, nesouvisí s funkcemi `IsWindow`, `IsWindowEnabled` a `IsWindowVisible` a jeho význam tedy z dokumentace není zřejmý. Experiment probíhá následovně:

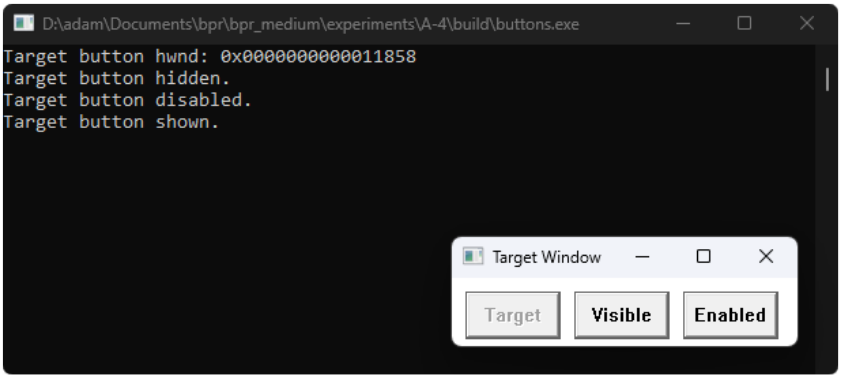
- Spustíme program `buttons.exe` (na obrázku A.2) s IL *high*, např. pomocí funkce *Run as administrator*.
- Pomocí tlačítek *Visible* resp. *Enabled* můžeme ovládat, zda je tlačítko *Target* viditelné resp. povolené.
- Vytvoříme demonstrační program `handle_validation.cpp`, který s *handle* tlačítka *Target* zavolá postupně funkce `IsWindow`, `IsWindowEnabled` a `IsWindowVisible`.

Můžeme pozorovat, že všechny funkce vrací hodnoty konzistentní s aktuálním stavem tlačítka *Target*. Za pozornost také stojí, že se dokumentace [193, 194, 195] nezmiňuje o tom, že by společně s těmito funkcemi měla být volána funkce `GetLastError`. To znamená, že Windows API nemá u těchto funkcí kanál, kterým by mohlo sdělit, že volání bylo blokováno UIPI. Proto bychom mohli spekulovat, že tyto funkce neimplementují kontrolu IL a do ochran poskytovaných UIPI nejsou začleněny.

## A.5 Chování filtru *window messages*

Cílem experimentů v této sekci je zmapovat chování filtru *window messages*, který zavádí UIPI. Klíčové je porovnání interakcí aplikací v případě, že je filtr v provozu, a v případě, že není. Dále se experimenty snaží nalézt anomálie v chování filtru a objasnit jejich vznik a důvody.

**Obrázek A.2** Rozhraní aplikace `buttons.exe` (obrázek autora)




Jak je rozebíráno v kapitole 4, pro aktivaci filtru *window messages* stačí, aby komunikující aplikace běžely na různém *integrity levelu*. V případě experimentů obsažených v této kapitole je použita konfigurace, kdy je `sender.exe` spuštěn s IL *medium* a `receiver.exe` s IL *high* prostřednictvím funkce *Run as administrator*. Nicméně, jak demonstrují experimenty A.5.4, chování filtru UIPI je totožné i v dalších konfiguracích, dokud je zachováno pravidlo, že `sender.exe` má nižší IL než `receiver.exe`. Podobným způsobem je v těchto experimentech ukázáno, že chování zůstane nezměněné i v případě, kdy filtr UIPI aktivní není, dokud jak `sender.exe`, tak `receiver.exe` běží na stejném IL.

V tabulkách v této sekci se návratovou hodnotou rozumí hodnota, kterou vrátila funkce `SendMessage`, a chybovým kódem se rozumí hodnota vracená funkci `GetLastError` bezprostředně po odeslání zprávy. Obě tyto hodnoty jsou uváděny ve své původní podobě, tak jak byly zapsány programem `receiver.exe`, nejedná se tedy o hodnoty získané následným manuálním testováním. Významy jednotlivých hodnot ve sloupečku „Stav“ jsou následující:

- blokovaná – zpráva měla chybový kód `0x5` již při automatizovaném testování;
- dodatečně blokovaná – zpráva měla chybový kód `0x5` až při manuálním testování;
- blokovaná `PostMessage` – zprávu se nepodařilo doručit pomocí `SendMessage`, ale při pokusu o odeslání pomocí `PostMessage` měla chybový kód `0x5`;
- nezprovozněná – zprávu se nepodařilo při manuálním testování doručit;
- doručená – zpráva byla doručena již při automatizovaném testování;
- dodatečně doručená (obecné) – zprávu se podařilo v manuálním testování doručit pomocí obecné úpravy popisované v A.5.3;
- dodatečně doručená (speciální) – zprávu se podařilo v manuálním testování doručit pomocí speciální úpravy popisované v A.5.3;
- doručená `PostMessage` – zprávu se nepodařilo doručit pomocí `SendMessage`, ale pomocí `PostMessage` se ji doručit podařilo;
- pouze `SendMessage` – zpráva měla chybový kód `0x487`, který indikuje, že ji nelze použít s `PostMessage` (viz. 4.4.2.2).

Důležité je poznamenat, že výpisy programů `sender.exe` a `receiver.exe` nezahrnují zprávy `WM_DESTROY`, `WM_CLOSE` a `WM_QUIT`. Jejich testování se provádí manuálně, neboť v případě, že na ně aplikace zareaguje, mohou narušit zbytek experimentu.

V tabulce A.2 lze nalézt přehled všech chybových kódů, které se ve výpisech testovacích programů uvedených v této práci vyskytují, včetně jejich odpovídající konstanty a chybové hlášky.

**Tabulka A.2** Přehled chybových kódů [33, 150, 151, 196]

| Hexadecimální číslice | Konstanta | Chybová hláška |
|---|---|---|
| 0x5 | ERROR_ACCESS_DENIED | *Access is denied.* |
| 0x6 | ERROR_INVALID_HANDLE | *The handle is invalid.* |
| 0x57 | ERROR_INVALID_PARAMETER | *The parameter is incorrect.* |
| 0x1e7 | ERROR_INVALID_ADDRESS | *Attempt to access invalid address.* |
| 0x3ea | ERROR_INVALID_MESSAGE | *The window cannot act on the sent message.* |
| 0x487 | ERROR_MESSAGE_SYNC_ONLY | *The message can be used only with synchronous operations.* |
| 0x578 | ERROR_INVALID_WINDOW_HANDLE | *Invalid window handle.* |
| 0x718 | ERROR_NOT_ENOUGH_QUOTA | *Not enough quota is available to process this command.* |

### A.5.1 Bez aktivovaného *User Interface Privilege Isolation*

První dvojice experimentů se zabývá situací, ve které se neaplikuje filtrování zavedené *User Interface Privilege Isolation*. Jak bylo rozebráno v kapitole 4 o UIPI, v případě, že obě aplikace, které komunikují, běží na stejném *integrity levelu*, omezení stanovená UIPI se jich netýkají.

#### A.5.1.1 Odesílání pomocí `SendMessage`

Experiment probíhá následovně:

- Aplikace `receiver.exe` a `sender.exe` spustíme na stejném IL, např. *medium*.
- Z aplikace `sender.exe` pošleme na okno aplikace `receiver.exe` všechny *window messages* v rozsahu `0x0` až `0xffff` s parametry `0`, `0` pomocí funkce `SendMessage`.
- Poté co dorazí poslední zpráva, vypneme aplikaci `receiver.exe` a z jejích logů odstraníme všechny záznamy, které předcházely spuštění aplikace `sernder.exe` a následovaly po jejím ukončení.

Porovnáme-li seznam zpráv, které byly odeslány programem `sender.exe` se seznamem zpráv, které skutečně dorazily do `WindowProc` programu `receiver.exe`, můžeme pozorovat, že z 65 536 odeslaných zpráv dorazilo 65 376. Zbývajících 160 zpráv doručeno nebylo. Tyto zprávy jsou dále analyzovány v sekci 4.4.1 a jejich přehled se nachází v tabulce A.3.

Dále výpis zpráv přijatých `WindowProc` programu `receiver.exe` obsahuje 34 zpráv, které nebyly programem `sender.exe` odeslány. Jejich přehled včetně důvodů, proč byly na `receiver.exe` doručeny se nachází v tabulce A.4.

**Tabulka A.3** Nedoručené *window messages* (`SendMessage` bez UIPI)

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_CREATE | 0x0 | 0x5 | blokovaná |
| WM_DEVMODECHANGE | 0x0 | 0x0 | dodatečně blokovaná |
| WM_GETMINMAXINFO | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_DRAWITEM | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_MEASUREITEM | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_DELETEITEM | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x38 | 0x0 | 0x0 | nezprovozněná |
| WM_COMPAREITEM | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x3a | 0x0 | 0x0 | doručená `PostMessage` |
| 0x3c | 0x0 | 0x0 | nezprovozněná |
| WM_WINDOWPOSCHANGING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_WINDOWPOSCHANGED | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_COPYGLOBALDATA | 0x0 | 0x0 | nezprovozněná |
| WM_COPYDATA | 0x0 | 0x0 | dodatečně doručená (speciální) |
| WM_NOTIFY | 0x0 | 0x5 | blokovaná |
| WM_HELP | 0x0 | 0x0 | dodatečně doručená (speciální) |
| 0x5a | 0x0 | 0x0 | nezprovozněná |
| 0x70 | 0x0 | 0x0 | nezprovozněná |
| WM_STYLECHANGING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_STYLECHANGED | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_NCCREATE | 0x0 | 0x0 | dodatečně blokovaná |
| WM_NCCALCSIZE | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x8b | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x8c | 0x0 | 0x0 | nezprovozněná |
| 0x91 | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x92 | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x93 | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x94 | 0x0 | 0x0 | dodatečně doručená (obecné) |

Tabulka pokračuje na další straně.

**Tabulka A.3** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| 0x95 | 0x0 | 0x0 | dodatečně doručená (obecné) |
| EM_SETRECT | 0x0 | 0x0 | dodatečně doručená (obecné) |
| EM_SETRECTNP | 0x0 | 0x0 | dodatečně doručená (obecné) |
| EM_GETLINE | 0x0 | 0x0 | nezprovozněná |
| SBM_SETSCROLLINFO | 0x0 | 0x0 | dodatečně doručená (obecné) |
| SBM_GETSCROLLINFO | 0x0 | 0x0 | dodatečně doručená (obecné) |
| SBM_GETSCROLLBARINFO | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_INITDIALOG | 0x0 | 0x5 | blokovaná |
| WM_GESTURE | 0x0 | 0x57 | nezprovozněná |
| WM_GESTURENOTIFY | 0x0 | 0x1e7 | dodatečně doručená (speciální) |
| 0x11b | 0x0 | 0x0 | nezprovozněná |
| 0x11c | 0x0 | 0x5 | blokovaná |
| WM_MENUGETOBJECT | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_ADDSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_DIR | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_GETLBTEXT | 0xfff…[1] | 0x0 | dodatečně doručená (obecné) |
| CB_INSERTSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_FINDSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_SELECTSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_FINDSTRINGEXACT | 0x0 | 0x0 | dodatečně doručená (obecné) |
| CB_GETCOMBOBOXINFO | 0x0 | 0x0 | dodatečně doručená (obecné) |
| LB_ADDSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| LB_INSERTSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| LB_GETTEXT | 0xfff…[1] | 0x0 | dodatečně doručená (obecné) |
| LB_SELECTSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |

Tabulka pokračuje na další straně.

[1] Zkráceno. Plná data jsou dostupná v souborech na přiloženém médiu.

**Tabulka A.3** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| LB_DIR | 0x0 | 0x0 | dodatečně doručená (obecné) |
| LB_FINDSTRING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| LB_ADDFILE | 0x0 | 0x0 | dodatečně doručená (obecné) |
| LB_GETITEMRECT | 0xfff…[2] | 0x0 | dodatečně doručená (obecné) |
| LB_FINDSTRINGEXACT | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_NEXTMENU | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_SIZING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_MOVING | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_MDICREATE | 0x0 | 0x0 | nezprovozněná |
| 0x22a | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x22b | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x22d | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x22e | 0x0 | 0x0 | dodatečně doručená (obecné) |
| 0x22f | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_POINTERDEVICECHANGE | 0x0 | 0x0 | doručená PostMessage |
| WM_POINTERDEVICEINRANGE | 0x0 | 0x0 | doručená PostMessage |
| WM_POINTERDEVICEOUTOFRANGE | 0x0 | 0x0 | doručená PostMessage |
| 0x23b | 0x0 | 0x0 | nezprovozněná |
| 0x23c | 0x0 | 0x0 | nezprovozněná |
| 0x23d | 0x0 | 0x0 | doručená PostMessage |
| WM_TOUCH | 0x0 | 0x57 | nezprovozněná |
| WM_NCPOINTERUPDATE | 0x0 | 0x0 | nezprovozněná |
| WM_NCPOINTERDOWN | 0x0 | 0x0 | nezprovozněná |
| WM_NCPOINTERUP | 0x0 | 0x0 | nezprovozněná |
| 0x244 | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERUPDATE | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERDOWN | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERUP | 0x0 | 0x0 | nezprovozněná |
| 0x248 | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERENTER | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERLEAVE | 0x0 | 0x0 | nezprovozněná |

Tabulka pokračuje na další straně.

[2] Zkráceno. Plná data jsou dostupná v souborech na přiloženém médiu.

**Tabulka A.3** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_POINTERACTIVATE | 0x0 | 0x57 | nezprovozněná |
| WM_POINTERCAPTURECHANGED | 0x0 | 0x0 | nezprovozněná |
| WM_TOUCHHITTESTING | 0x0 | 0x1e7 | dodatečně doručená (speciální) |
| WM_POINTERWHEEL | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERHWHEEL | 0x0 | 0x0 | nezprovozněná |
| DM_POINTERHITTEST | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERROUTEDTO | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERROUTEDAWAY | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERROUTEDRELEASED | 0x0 | 0x0 | nezprovozněná |
| 0x254 | 0x0 | 0x0 | nezprovozněná |
| 0x255 | 0x0 | 0x0 | nezprovozněná |
| 0x256 | 0x0 | 0x0 | nezprovozněná |
| 0x257 | 0x0 | 0x0 | nezprovozněná |
| WM_TABLET_ADDED | 0x0 | 0x0 | doručená PostMessage |
| WM_TABLET_DELETED | 0x0 | 0x0 | doručená PostMessage |
| 0x2cd | 0x0 | 0x0 | nezprovozněná |
| WM_DPICHANGED | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_GETDPISCALEDSIZE | 0x0 | 0x0 | dodatečně doručená (speciální) |
| 0x2e6 | 0x0 | 0x0 | blokovaná PostMessage |
| 0x2fb | 0x0 | 0x0 | blokovaná PostMessage |
| WM_PAINTCLIPBOARD | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_SIZECLIPBOARD | 0x0 | 0x0 | dodatečně doručená (obecné) |
| WM_ASKCBFORMATNAME | 0x0 | 0x57 | dodatečně doručená (speciální) |
| 0x314 | 0x0 | 0x0 | nezprovozněná |
| 0x324 | 0x0 | 0x0 | doručená PostMessage |
| 0x325 | 0x0 | 0x0 | doručená PostMessage |
| 0x329 | 0x0 | 0x0 | doručená PostMessage |
| 0x32c | 0x0 | 0x0 | blokovaná PostMessage |
| 0x32d | 0x0 | 0x0 | blokovaná PostMessage |
| 0x32e | 0x0 | 0x0 | blokovaná PostMessage |
| 0x32f | 0x0 | 0x0 | blokovaná PostMessage |
| WM_GETTITLEBARINFOEX | 0x0 | 0x0 | dodatečně doručená (speciální) |
| 0x341 | 0x0 | 0x5 | blokovaná |
| 0x342 | 0x0 | 0x5 | blokovaná |
| 0x343 | 0x0 | 0x0 | blokovaná PostMessage |
| 0x344 | 0x0 | 0x5 | blokovaná |
| WM_TOOLTIPDISMISS | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_FIRST/WM_DDE_INITIATE | 0x0 | 0x578 | dodatečně doručená (speciální) |

Tabulka pokračuje na další straně.

**Tabulka A.3** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_DDE_TERMINATE | 0x0 | 0x0 | nezprovozněná |
| WM_DDE_ADVISE | 0x0 | 0x0 | nezprovozněná |
| WM_DDE_UNADVISE | 0x0 | 0x0 | nezprovozněná |
| WM_DDE_ACK | 0x0 | 0x578 | dodatečně doručená (speciální) |
| WM_DDE_DATA | 0x0 | 0x0 | nezprovozněná |
| WM_DDE_REQUEST | 0x0 | 0x0 | nezprovozněná |
| WM_DDE_POKE | 0x0 | 0x0 | nezprovozněná |
| WM_DDE_EXECUTE/WM_DDE_LAST | 0x0 | 0x0 | nezprovozněná |

**Tabulka A.4** Neodeslané *window messages* (SendMessage bez UIPI)

| Zpráva | wParam | lParam | Vysvětlení |
|---|---|---|---|
| WM_ENDSESSION | 0x0 | 0x0 | generovaná 0x3b |
| WM_HELP | 0x0 | 0xb5c…[3] | generovaná 0x4d |
| WM_GETICON | 0x2 | 0x0 | nesouvisející |
| WM_GETICON | 0x0 | 0x0 | nesouvisející |
| WM_GETICON | 0x1 | 0x0 | nesouvisející |
| WM_NCMOUSELEAVE | 0x0 | 0x0 | generovaná WM_NCMOUSEMOVE |
| CB_GETLBTEXTLEN | 0x0 | 0x0 | generovaná CB_GETLBTEXT |
| LB_GETTEXTLEN | 0x0 | 0x0 | generovaná LB_GETTEXT |
| WM_CONTEXTMENU | 0x230…[3] | 0x87f…[3] | generovaná WM_RBUTTONUP |
| WM_IME_NOTIFY | 0x0 | 0x0 | generovaná WM_IME_SETCONTEXT |
| WM_CHAR | 0x0 | 0x1 | generovaná WM_IME_CHAR |
| WM_KEYDOWN/WM_KEYFIRST | 0x0 | 0x0 | generovaná WM_IME_KEYDOWN |
| WM_KEYUP | 0x0 | 0x0 | generovaná WM_IME_KEYUP |
| WM_NCHITTEST | 0x0 | 0x0 | generovaná WM_TABLET_QUERYSYSTEMGESTURESTATUS |
| WM_WINDOWPOSCHANGING | 0x0 | 0xb5c…[3] | generovaná 0x313 |
| WM_ACTIVATEAPP | 0x1 | 0x0 | generovaná 0x313 |
| WM_NCACTIVATE | 0x0 | 0x0 | generovaná 0x313 |
| WM_ACTIVATE | 0x1 | 0x0 | generovaná 0x313 |
| WM_IME_SETCONTEXT | 0x1 | 0xc00…[3] | generovaná 0x313 |
| WM_IME_NOTIFY | 0x2 | 0x0 | generovaná 0x313 |
| WM_GETOBJECT | 0xfff…[3] | 0xfff…[3] | generovaná 0x313 |
| WM_GETOBJECT | 0x0 | 0xfff…[3] | generovaná 0x313 |
| WM_SETFOCUS | 0x0 | 0x0 | generovaná 0x313 |
| WM_WINDOWPOSCHANGING | 0x0 | 0xb5c…[3] | generovaná WM_DWMCOMPOSITIONCHANGED |
| WM_NCCALCSIZE | 0x1 | 0xb5c…[3] | generovaná WM_DWMCOMPOSITIONCHANGED |

Tabulka pokračuje na další straně.

[3]Zkráceno. Plná data jsou dostupná v souborech na přiloženém médiu.

**Tabulka A.4** – pokračování z předchozí strany

| Zpráva | wParam | lParam | Vysvětlení |
|---|---|---|---|
| WM_WINDOWPOSCHANGED | 0x0 | 0xb5c…[4] | generovaná WM_DWMCOMPOSITIONCHANGED |
| WM_GETICON | 0x2 | 0x0 | nesouvisející |
| WM_GETICON | 0x0 | 0x0 | nesouvisející |
| WM_GETICON | 0x1 | 0x0 | nesouvisející |
| WM_WINDOWPOSCHANGING | 0x0 | 0xb5c…[4] | generovaná WM_DWMNCRENDERINGCHANGED |
| WM_NCCALCSIZE | 0x1 | 0xb5c…[4] | generovaná WM_DWMNCRENDERINGCHANGED |
| WM_WINDOWPOSCHANGED | 0x0 | 0xb5c…[4] | generovaná WM_DWMNCRENDERINGCHANGED |
| WM_NCCALCSIZE | 0x1 | 0xb5c…[4] | generovaná WM_DWMNCRENDERINGCHANGED |

#### A.5.1.2 Odesílání pomocí PostMessage

Experiment probíhá následovně:

- Aplikace `receiver.exe` a `sender.exe` spustíme na stejném IL, např. *medium*.
- Z aplikace `sender.exe` pošleme na okno aplikace `receiver.exe` všechny *window messages* v rozsahu `0x0` až `0xffff` s parametry `0`, `0` pomocí funkce `PostMessage`.
- Poté co dorazí poslední zpráva, vypneme aplikaci `receiver.exe` a z jejích logů odstraníme všechny záznamy, které předcházely spuštění aplikace `sender.exe` a následovaly po jejím ukončení.

V první řadě narazíme na problém s tím, že po odeslání prvních zhruba 14 000 *window messages* začne `sender.exe` u některých zpráv hlásit chybový kód `0x718`. Ten značí, že cílový proces je zahlcen a nestíhá zprávy zpracovávat, konkrétně se překládá na chybovou hlášku „*Not enough quota is available to process this command,*“ [196]. K přehlcení pravděpodobně dochází z toho důvodu, že každá přijatá *window message* je vypisována na výstup hned dvakrát, jednou za `wWinMain` a podruhé za `WindowProc`. Řešením je rozdělit experiment do několika fází po cca 14 000 zprávách. Výsledky tohoto experimentu jsou složeny konkrétně z pěti fází s předěly v `0x3771`, `0x6dd6`, `0xa3e2` a `0xd7cf`.

Složíme-li výsledky všech experimentů v jednu sadu souborů a porovnáme-li seznam zpráv, které byly odeslány programem `sender.exe`, se seznamem zpráv, které skutečně dorazily do `wWinMain` programu `receiver.exe`, můžeme pozorovat, že z 65 536 odeslaných zpráv dorazilo 65 367. Zbývajících 169 zpráv doručeno nebylo. Tyto zprávy jsou dále analyzovány v sekci 4.4.2 a jejich přehled se nachází v tabulce A.5.

Výpis zpráv, které byly zpracovány ve `wWinMain` programu `receiver.exe`, obsahuje navíc 4 zprávy, které nebyly odeslány programem `sender.exe`. Oproti experimentu se `SendMessage` je ale každá z nich vedlejším výsledkem dalšího dění na počítači, na kterém experiment probíhal, a nejsou tak signifikantní pro výsledek experimentu. I přesto je jejich přehled uveden v tabulce A.6.

[4] Zkráceno. Plná data jsou dostupná v souborech na přiloženém médiu.

**Tabulka A.5** Nedoručené *window messages* (`PostMessage` bez UIPI)

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_CREATE | 0x0 | 0x487 | pouze SendMessage |
| WM_SETTEXT | 0x0 | 0x487 | pouze SendMessage |
| WM_GETTEXT | 0x0 | 0x487 | pouze SendMessage |
| WM_GETTEXTLENGTH | 0x0 | 0x487 | pouze SendMessage |
| WM_ERASEBKGND | 0x0 | 0x487 | pouze SendMessage |
| WM_WININICHANGE | 0x0 | 0x487 | pouze SendMessage |
| WM_DEVMODECHANGE | 0x0 | 0x487 | pouze SendMessage |
| WM_GETMINMAXINFO | 0x0 | 0x487 | pouze SendMessage |
| WM_ICONERASEBKGND | 0x0 | 0x487 | pouze SendMessage |
| WM_DRAWITEM | 0x0 | 0x487 | pouze SendMessage |
| WM_MEASUREITEM | 0x0 | 0x487 | pouze SendMessage |
| WM_DELETEITEM | 0x0 | 0x487 | pouze SendMessage |
| WM_GETFONT | 0x0 | 0x487 | pouze SendMessage |
| 0x38 | 0x0 | 0x487 | pouze SendMessage |
| WM_COMPAREITEM | 0x0 | 0x487 | pouze SendMessage |
| 0x3c | 0x0 | 0x487 | pouze SendMessage |
| WM_WINDOWPOSCHANGING | 0x0 | 0x487 | pouze SendMessage |
| WM_WINDOWPOSCHANGED | 0x0 | 0x487 | pouze SendMessage |
| WM_COPYGLOBALDATA | 0x0 | 0x487 | pouze SendMessage |
| WM_COPYDATA | 0x0 | 0x487 | pouze SendMessage |
| WM_NOTIFY | 0x0 | 0x5 | blokovaná |
| WM_HELP | 0x0 | 0x487 | pouze SendMessage |
| 0x59 | 0x0 | 0x487 | pouze SendMessage |
| 0x5a | 0x0 | 0x487 | pouze SendMessage |
| 0x70 | 0x0 | 0x487 | pouze SendMessage |
| WM_STYLECHANGING | 0x0 | 0x487 | pouze SendMessage |
| WM_STYLECHANGED | 0x0 | 0x487 | pouze SendMessage |
| WM_NCCREATE | 0x0 | 0x487 | pouze SendMessage |
| WM_NCCALCSIZE | 0x0 | 0x487 | pouze SendMessage |
| WM_NCPAINT | 0x0 | 0x487 | pouze SendMessage |
| WM_GETDLGCODE | 0x0 | 0x487 | pouze SendMessage |
| 0x8b | 0x0 | 0x487 | pouze SendMessage |
| 0x8c | 0x0 | 0x487 | pouze SendMessage |
| 0x90 | 0x0 | 0x487 | pouze SendMessage |
| 0x91 | 0x0 | 0x487 | pouze SendMessage |
| 0x92 | 0x0 | 0x487 | pouze SendMessage |
| 0x93 | 0x0 | 0x487 | pouze SendMessage |
| 0x94 | 0x0 | 0x487 | pouze SendMessage |
| 0x95 | 0x0 | 0x487 | pouze SendMessage |
| EM_GETSEL | 0x0 | 0x487 | pouze SendMessage |
| EM_GETRECT | 0x0 | 0x487 | pouze SendMessage |
| EM_SETRECT | 0x0 | 0x487 | pouze SendMessage |
| EM_SETRECTNP | 0x0 | 0x487 | pouze SendMessage |
| EM_REPLACESEL | 0x0 | 0x487 | pouze SendMessage |
| EM_GETLINE | 0x0 | 0x487 | pouze SendMessage |
| EM_SETTABSTOPS | 0x0 | 0x487 | pouze SendMessage |

Tabulka pokračuje na další straně.

**Tabulka A.5** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| SBM_GETRANGE | 0x0 | 0x487 | pouze SendMessage |
| SBM_SETSCROLLINFO | 0x0 | 0x487 | pouze SendMessage |
| SBM_GETSCROLLINFO | 0x0 | 0x487 | pouze SendMessage |
| SBM_GETSCROLLBARINFO | 0x0 | 0x487 | pouze SendMessage |
| WM_KEYLAST | 0x0 | 0x487 | pouze SendMessage |
| WM_CONVERTREQUEST | 0x0 | 0x487 | pouze SendMessage |
| WM_INITDIALOG | 0x0 | 0x487 | pouze SendMessage |
| WM_GESTURE | 0x0 | 0x57 | nezprovozněná |
| WM_GESTURENOTIFY | 0x0 | 0x487 | pouze SendMessage |
| 0x11b | 0x0 | 0x3ea | nezprovozněná |
| 0x11c | 0x0 | 0x487 | pouze SendMessage |
| WM_MENUGETOBJECT | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLORMSGBOX | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLOREDIT | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLORLISTBOX | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLORBTN | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLORDLG | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLORSCROLLBAR | 0x0 | 0x487 | pouze SendMessage |
| WM_CTLCOLORSTATIC | 0x0 | 0x487 | pouze SendMessage |
| CB_GETEDITSEL | 0x0 | 0x487 | pouze SendMessage |
| CB_ADDSTRING | 0x0 | 0x487 | pouze SendMessage |
| CB_GETLBTEXT | 0x0 | 0x487 | pouze SendMessage |
| CB_GETLBTEXTLEN | 0x0 | 0x487 | pouze SendMessage |
| CB_INSERTSTRING | 0x0 | 0x487 | pouze SendMessage |
| CB_FINDSTRING | 0x0 | 0x487 | pouze SendMessage |
| CB_SELECTSTRING | 0x0 | 0x487 | pouze SendMessage |
| CB_GETDROPPEDCONTROLRECT | 0x0 | 0x487 | pouze SendMessage |
| CB_FINDSTRINGEXACT | 0x0 | 0x487 | pouze SendMessage |
| CB_GETCOMBOBOXINFO | 0x0 | 0x487 | pouze SendMessage |
| LB_ADDSTRING | 0x0 | 0x487 | pouze SendMessage |
| LB_INSERTSTRING | 0x0 | 0x487 | pouze SendMessage |
| LB_GETTEXT | 0x0 | 0x487 | pouze SendMessage |
| LB_GETTEXTLEN | 0x0 | 0x487 | pouze SendMessage |
| LB_SELECTSTRING | 0x0 | 0x487 | pouze SendMessage |
| LB_FINDSTRING | 0x0 | 0x487 | pouze SendMessage |
| LB_GETSELITEMS | 0x0 | 0x487 | pouze SendMessage |
| LB_SETTABSTOPS | 0x0 | 0x487 | pouze SendMessage |
| LB_ADDFILE | 0x0 | 0x487 | pouze SendMessage |
| LB_GETITEMRECT | 0x0 | 0x487 | pouze SendMessage |
| LB_FINDSTRINGEXACT | 0x0 | 0x487 | pouze SendMessage |
| 0x1aa | 0x0 | 0x487 | pouze SendMessage |
| 0x1ab | 0x0 | 0x487 | pouze SendMessage |
| 0x1ac | 0x0 | 0x487 | pouze SendMessage |
| 0x1ad | 0x0 | 0x487 | pouze SendMessage |
| LB_GETLISTBOXINFO | 0x0 | 0x487 | pouze SendMessage |

Tabulka pokračuje na další straně.

**Tabulka A.5** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| 0x1eb | 0x0 | 0x487 | pouze SendMessage |
| WM_PARENTNOTIFY | 0x0 | 0x487 | pouze SendMessage |
| WM_NEXTMENU | 0x0 | 0x487 | pouze SendMessage |
| WM_SIZING | 0x0 | 0x487 | pouze SendMessage |
| WM_MOVING | 0x0 | 0x487 | pouze SendMessage |
| WM_MDICREATE | 0x0 | 0x487 | pouze SendMessage |
| WM_MDIGETACTIVE | 0x0 | 0x487 | pouze SendMessage |
| 0x22a | 0x0 | 0x487 | pouze SendMessage |
| 0x22b | 0x0 | 0x487 | pouze SendMessage |
| 0x22d | 0x0 | 0x487 | pouze SendMessage |
| 0x22e | 0x0 | 0x487 | pouze SendMessage |
| 0x22f | 0x0 | 0x487 | pouze SendMessage |
| WM_DROPFILES | 0x0 | 0x6 | nezprovozněná |
| 0x23b | 0x1 | 0x0 | nezprovozněná |
| 0x23c | 0x1 | 0x0 | nezprovozněná |
| WM_TOUCH | 0x0 | 0x57 | nezprovozněná |
| WM_NCPOINTERUPDATE | 0x0 | 0x3ea | nezprovozněná |
| WM_NCPOINTERDOWN | 0x0 | 0x3ea | nezprovozněná |
| WM_NCPOINTERUP | 0x0 | 0x3ea | nezprovozněná |
| 0x244 | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERUPDATE | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERDOWN | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERUP | 0x0 | 0x3ea | nezprovozněná |
| 0x248 | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERENTER | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERLEAVE | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERACTIVATE | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERCAPTURECHANGED | 0x0 | 0x3ea | nezprovozněná |
| WM_TOUCHHITTESTING | 0x0 | 0x487 | pouze SendMessage |
| WM_POINTERWHEEL | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERHWHEEL | 0x0 | 0x3ea | nezprovozněná |
| DM_POINTERHITTEST | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERROUTEDTO | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERROUTEDAWAY | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERROUTEDRELEASED | 0x0 | 0x3ea | nezprovozněná |
| 0x254 | 0x0 | 0x3ea | nezprovozněná |
| 0x255 | 0x0 | 0x3ea | nezprovozněná |
| 0x256 | 0x0 | 0x3ea | nezprovozněná |
| 0x257 | 0x0 | 0x3ea | nezprovozněná |
| WM_IME_SETCONTEXT | 0x0 | 0x487 | pouze SendMessage |
| WM_IME_CONTROL | 0x0 | 0x487 | pouze SendMessage |
| WM_IME_REQUEST | 0x0 | 0x487 | pouze SendMessage |
| 0x2cd | 0x0 | 0x0 | nezprovozněná |
| WM_DPICHANGED | 0x0 | 0x487 | pouze SendMessage |
| 0x2e1 | 0x0 | 0x487 | pouze SendMessage |

Tabulka pokračuje na další straně.

**Tabulka A.5** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_DPICHANGED_BEFOREPARENT | 0x0 | 0x487 | pouze SendMessage |
| WM_DPICHANGED_AFTERPARENT | 0x0 | 0x487 | pouze SendMessage |
| WM_GETDPISCALEDSIZE | 0x0 | 0x487 | pouze SendMessage |
| 0x2e5 | 0x0 | 0x487 | pouze SendMessage |
| 0x2e6 | 0x0 | 0x5 | blokovaná |
| 0x2ed | 0x0 | 0x5 | blokovaná |
| 0x2ee | 0x0 | 0x5 | blokovaná |
| 0x2fa | 0x0 | 0x487 | pouze SendMessage |
| 0x2fb | 0x0 | 0x5 | blokovaná |
| WM_PAINTCLIPBOARD | 0x0 | 0x487 | pouze SendMessage |
| WM_SIZECLIPBOARD | 0x0 | 0x487 | pouze SendMessage |
| WM_ASKCBFORMATNAME | 0x0 | 0x487 | pouze SendMessage |
| 0x314 | 0x0 | 0x487 | pouze SendMessage |
| 0x32c | 0x0 | 0x5 | blokovaná |
| 0x32d | 0x0 | 0x5 | blokovaná |
| 0x32e | 0x0 | 0x5 | blokovaná |
| 0x32f | 0x0 | 0x5 | blokovaná |
| WM_GETTITLEBARINFOEX | 0x0 | 0x487 | pouze SendMessage |
| 0x341 | 0x0 | 0x487 | pouze SendMessage |
| 0x342 | 0x0 | 0x5 | blokovaná |
| 0x343 | 0x0 | 0x5 | blokovaná |
| 0x344 | 0x0 | 0x5 | blokovaná |
| WM_TOOLTIPDISMISS | 0x0 | 0x5 | blokovaná |
| 0x348 | 0x0 | 0x5 | blokovaná |
| 0x349 | 0x0 | 0x5 | blokovaná |
| WM_DDE_FIRST/WM_DDE_INITIATE | 0x0 | 0x578 | nezprovozněná |
| WM_DDE_ADVISE | 0x1 | 0x578 | nezprovozněná |
| WM_DDE_UNADVISE | 0x1 | 0x578 | nezprovozněná |
| WM_DDE_ACK | 0x1 | 0x578 | nezprovozněná |
| WM_DDE_DATA | 0x1 | 0x578 | nezprovozněná |
| WM_DDE_REQUEST | 0x1 | 0x578 | nezprovozněná |
| WM_DDE_POKE | 0x1 | 0x578 | nezprovozněná |
| WM_DDE_EXECUTE/WM_DDE_LAST | 0x1 | 0x578 | nezprovozněná |

**Tabulka A.6** Neodeslané *window messages* (PostMessage bez UIPI)

| Zpráva | wParam | lParam | Vysvětlení |
|---|---|---|---|
| WM_NCMOUSELEAVE | 0x0 | 0x0 | nesouvisející |
| WM_CHAR | 0x0 | 0x1 | nesouvisející |
| WM_KEYDOWN/WM_KEYFIRST | 0x0 | 0x0 | nesouvisející |
| WM_KEYUP | 0x0 | 0x0 | nesouvisející |

## A.5.2 S aktivovaným *User Interface Privilege Isolation*

Druhá dvojice experimentů se od předchozích experimentů s filtrem *window messages* liší v tom, že zkoumá situaci, kdy je filtr v provozu. K tomu dochází, jakmile obě aplikace běží na různých *integrity levelech*, přičemž přijímající aplikace `receiver.exe` má IL vyšší než odesílající aplikace `sender.exe`.

### A.5.2.1 Odesílání pomocí `SendMessage`

Experiment probíhá následovně:

- Aplikaci `receiver.exe` spustíme na IL *high* prostřednictvím funkce *Run as administrator* a aplikaci `sender.exe` spustíme na IL *medium*.
- Z aplikace `sender.exe` pošleme na okno aplikace `receiver.exe` všechny *window messages* v rozsahu `0x0` až `0xffff` s parametry `0`, `0` pomocí funkce `SendMessage`.
- Poté co dorazí poslední zpráva, vypneme aplikaci `receiver.exe` a z jejích logů odstraníme všechny záznamy, které předcházely spuštění aplikace `sender.exe` a následovaly po jejím ukončení.

Můžeme pozorovat, že do `WindowProc` programu `receiver.exe` dorazí 27 zpráv, přičemž 11 z nich nebylo odesláno `sender.exe`. Celkem 16 zpráv tedy při automatizovaném testování prošlo UIPI filtrem. Nicméně dalších 107 zpráv v aplikaci `sender.exe` buď indikovalo chybu jinou než `0x5`, nebo chybu neindikovalo, ale přesto nedošlo k jejich doručení. Výsledky experimentu včetně výsledků manuálního testování, kterému bylo podrobeno těchto 107 zpráv, jsou rozebírány v sekci 4.4.3. V tabulce A.7 se pak nachází přehled všech *window messages*, které nebyly explicitně blokovány.

**Tabulka A.7** Jiné než blokované *window messages* (`SendMessage` s UIPI)

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| `WM_NULL` | `0x0` | `0x0` | doručená |
| `WM_MOVE` | `0x0` | `0x0` | doručená |
| `WM_SIZE` | `0x0` | `0x0` | doručená |
| `WM_GETTEXT` | `0x0` | `0x0` | doručená |
| `WM_GETTEXTLENGTH` | `0xd` | `0x0` | doručená |
| `WM_GETMINMAXINFO` | `0x0` | `0x0` | dodatečně blokovaná |
| `WM_DRAWITEM` | `0x0` | `0x0` | dodatečně blokovaná |
| `WM_MEASUREITEM` | `0x0` | `0x0` | dodatečně blokovaná |
| `WM_DELETEITEM` | `0x0` | `0x0` | dodatečně blokovaná |
| `WM_GETHOTKEY` | `0x0` | `0x0` | doručená |
| `0x38` | `0x0` | `0x0` | nezprovozněná |
| `WM_COMPAREITEM` | `0x0` | `0x0` | dodatečně blokovaná |
| `0x3a` | `0x0` | `0x0` | blokovaná `PostMessage` |
| `0x3c` | `0x0` | `0x0` | blokovaná `PostMessage` |
| `WM_WINDOWPOSCHANGING` | `0x0` | `0x0` | dodatečně blokovaná |
| `WM_WINDOWPOSCHANGED` | `0x0` | `0x0` | dodatečně blokovaná |
| `WM_COPYGLOBALDATA` | `0x0` | `0x0` | blokovaná `PostMessage` |
| `WM_COPYDATA` | `0x0` | `0x0` | dodatečně blokovaná |

Tabulka pokračuje na další straně.

**Tabulka A.7** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_HELP | 0x0 | 0x0 | dodatečně blokovaná |
| 0x5a | 0x0 | 0x0 | blokovaná PostMessage |
| 0x70 | 0x0 | 0x0 | blokovaná PostMessage |
| WM_STYLECHANGING | 0x0 | 0x0 | dodatečně blokovaná |
| WM_STYLECHANGED | 0x0 | 0x0 | dodatečně blokovaná |
| WM_GETICON | 0x0 | 0x0 | doručená |
| WM_NCCREATE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_NCCALCSIZE | 0x0 | 0x0 | dodatečně blokovaná |
| 0x8b | 0x0 | 0x0 | dodatečně blokovaná |
| 0x8c | 0x0 | 0x0 | blokovaná PostMessage |
| 0x91 | 0x0 | 0x0 | dodatečně blokovaná |
| 0x92 | 0x0 | 0x0 | dodatečně blokovaná |
| 0x93 | 0x0 | 0x0 | dodatečně blokovaná |
| 0x94 | 0x0 | 0x0 | dodatečně blokovaná |
| 0x95 | 0x0 | 0x0 | dodatečně blokovaná |
| EM_SETRECT | 0x0 | 0x0 | dodatečně blokovaná |
| EM_SETRECTNP | 0x0 | 0x0 | dodatečně blokovaná |
| EM_GETLINE | 0x0 | 0x0 | blokovaná PostMessage |
| SBM_SETSCROLLINFO | 0x0 | 0x0 | dodatečně blokovaná |
| SBM_GETSCROLLINFO | 0x0 | 0x0 | dodatečně blokovaná |
| SBM_GETSCROLLBARINFO | 0x0 | 0x0 | dodatečně blokovaná |
| WM_GESTURE | 0x0 | 0x57 | nezprovozněná |
| WM_GESTURENOTIFY | 0x0 | 0x1e7 | dodatečně blokovaná |
| 0x11b | 0x0 | 0x0 | nezprovozněná |
| WM_MENUGETOBJECT | 0x0 | 0x0 | dodatečně blokovaná |
| CB_GETCOMBOBOXINFO | 0x0 | 0x0 | dodatečně blokovaná |
| LB_GETITEMRECT | 0xfff…[5] | 0x0 | dodatečně blokovaná |
| WM_NEXTMENU | 0x0 | 0x0 | dodatečně blokovaná |
| WM_SIZING | 0x0 | 0x0 | dodatečně blokovaná |
| WM_MOVING | 0x0 | 0x0 | dodatečně blokovaná |
| WM_MDICREATE | 0x0 | 0x0 | blokovaná PostMessage |
| 0x22a | 0x0 | 0x0 | dodatečně blokovaná |
| 0x22b | 0x0 | 0x0 | dodatečně blokovaná |
| 0x22d | 0x0 | 0x0 | dodatečně blokovaná |
| 0x22e | 0x0 | 0x0 | dodatečně blokovaná |
| 0x22f | 0x0 | 0x0 | dodatečně blokovaná |
| WM_POINTERDEVICECHANGE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_POINTERDEVICEINRANGE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_POINTERDEVICEOUTOFRANGE | 0x0 | 0x0 | blokovaná PostMessage |
| 0x23b | 0x0 | 0x0 | blokovaná PostMessage |
| 0x23c | 0x0 | 0x0 | blokovaná PostMessage |
| 0x23d | 0x0 | 0x0 | blokovaná PostMessage |
| WM_TOUCH | 0x0 | 0x57 | nezprovozněná |
| WM_NCPOINTERUPDATE | 0x0 | 0x0 | nezprovozněná |

Tabulka pokračuje na další straně.

[5] Zkráceno. Plná data jsou dostupná v souborech na přiloženém médiu.

**Tabulka A.7** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_NCPOINTERDOWN | 0x0 | 0x0 | nezprovozněná |
| WM_NCPOINTERUP | 0x0 | 0x0 | nezprovozněná |
| 0x244 | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERUPDATE | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERDOWN | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERUP | 0x0 | 0x0 | nezprovozněná |
| 0x248 | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERENTER | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERLEAVE | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERACTIVATE | 0x0 | 0x57 | nezprovozněná |
| WM_POINTERCAPTURECHANGED | 0x0 | 0x0 | nezprovozněná |
| WM_TOUCHHITTESTING | 0x0 | 0x1e7 | dodatečně blokovaná |
| WM_POINTERWHEEL | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERHWHEEL | 0x0 | 0x0 | nezprovozněná |
| DM_POINTERHITTEST | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERROUTEDTO | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERROUTEDAWAY | 0x0 | 0x0 | nezprovozněná |
| WM_POINTERROUTEDRELEASED | 0x0 | 0x0 | nezprovozněná |
| 0x254 | 0x0 | 0x0 | nezprovozněná |
| 0x255 | 0x0 | 0x0 | nezprovozněná |
| 0x256 | 0x0 | 0x0 | nezprovozněná |
| 0x257 | 0x0 | 0x0 | nezprovozněná |
| WM_TABLET_ADDED | 0x0 | 0x0 | blokovaná PostMessage |
| WM_TABLET_DELETED | 0x0 | 0x0 | blokovaná PostMessage |
| 0x2cd | 0x0 | 0x0 | nezprovozněná |
| WM_DPICHANGED | 0x0 | 0x0 | dodatečně blokovaná |
| WM_GETDPISCALEDSIZE | 0x0 | 0x0 | dodatečně blokovaná |
| 0x2e6 | 0x0 | 0x0 | blokovaná PostMessage |
| 0x2fb | 0x0 | 0x0 | blokovaná PostMessage |
| WM_RENDERFORMAT | 0x0 | 0x0 | doručená |
| WM_DRAWCLIPBOARD | 0x0 | 0x0 | doručená |
| WM_PAINTCLIPBOARD | 0x0 | 0x0 | dodatečně blokovaná |
| WM_SIZECLIPBOARD | 0x0 | 0x0 | dodatečně blokovaná |
| WM_ASKCBFORMATNAME | 0x0 | 0x57 | dodatečně blokovaná |
| WM_CHANGECBCHAIN | 0x0 | 0x0 | doručená |
| 0x313 | 0x0 | 0x0 | doručená |
| 0x314 | 0x0 | 0x0 | blokovaná PostMessage |
| WM_THEMECHANGED | 0x0 | 0x0 | doručená |
| 0x31b | 0x1 | 0x0 | doručená |
| WM_DWMNCRENDERINGCHANGED | 0x0 | 0x0 | doručená |
| 0x324 | 0x0 | 0x0 | blokovaná PostMessage |
| 0x325 | 0x0 | 0x0 | blokovaná PostMessage |
| 0x329 | 0x0 | 0x0 | blokovaná PostMessage |
| 0x32c | 0x0 | 0x0 | blokovaná PostMessage |
| 0x32d | 0x0 | 0x0 | blokovaná PostMessage |

Tabulka pokračuje na další straně.

**Tabulka A.7** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| 0x32e | 0x0 | 0x0 | blokovaná PostMessage |
| 0x32f | 0x0 | 0x0 | blokovaná PostMessage |
| WM_GETTITLEBARINFOEX | 0x0 | 0x0 | dodatečně blokovaná |
| 0x343 | 0x0 | 0x0 | blokovaná PostMessage |
| WM_TOOLTIPDISMISS | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_FIRST/WM_DDE_INITIATE | 0x0 | 0x578 | blokovaná PostMessage |
| WM_DDE_TERMINATE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_ADVISE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_UNADVISE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_ACK | 0x0 | 0x578 | blokovaná PostMessage |
| WM_DDE_DATA | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_REQUEST | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_POKE | 0x0 | 0x0 | blokovaná PostMessage |
| WM_DDE_EXECUTE/WM_DDE_LAST | 0x0 | 0x0 | blokovaná PostMessage |
| 0xc0a1 | 0x0 | 0x0 | doručená |
| 0xc0a2 | 0x0 | 0x0 | doručená |

**Tabulka A.8** Neodeslané *window messages* (SendMessage s UIPI)

| Zpráva | wParam | lParam | Vysvětlení |
|---|---|---|---|
| WM_WINDOWPOSCHANGING | 0x0 | 0x85be0fefb0 | generovaná 0x313 |
| WM_ACTIVATEAPP | 0x1 | 0x0 | generovaná 0x313 |
| WM_NCACTIVATE | 0x200000 | 0x0 | generovaná 0x313 |
| WM_ACTIVATE | 0x200001 | 0x0 | generovaná 0x313 |
| WM_WINDOWPOSCHANGING | 0x0 | 0x85be0ff0c0 | generovaná WM_DWMNCRENDERINGCHANGED |
| WM_NCCALCSIZE | 0x1 | 0x85be0ff090 | generovaná WM_DWMNCRENDERINGCHANGED |
| WM_WINDOWPOSCHANGED | 0x0 | 0x85be0ff0c0 | generovaná WM_DWMNCRENDERINGCHANGED |
| WM_GETICON | 0x2 | 0x0 | nesouvisející |
| WM_GETICON | 0x0 | 0x0 | nesouvisející |
| WM_GETICON | 0x1 | 0x0 | nesouvisející |
| WM_NCCALCSIZE | 0x1 | 0x85be0fe890 | generovaná WM_DWMNCRENDERINGCHANGED |

### A.5.2.2 Odesílání pomocí PostMessage

Experiment probíhá následovně:

- Aplikaci `receiver.exe` spustíme na IL *high* prostřednictvím funkce *Run as administrator* a aplikaci `sender.exe` spustíme na IL *medium*.

- Z aplikace `sender.exe` pošleme na okno aplikace `receiver.exe` všechny *window messages* v rozsahu `0x0` až `0xffff` s parametry `0`, `0` pomocí funkce `PostMessage`.
- Poté co dorazí poslední zpráva, vypneme aplikaci `receiver.exe` a z jejích logů odstraníme všechny záznamy, které předcházely spuštění aplikace `sender.exe` a následovaly po jejím ukončení.

Můžeme pozorovat, že `receiver.exe` ve své `wWinMain` zaznamenal celkem 14 příchozích zpráv, které při automatizovaném testování prošly filtrem. U dalších 29 *window messages* můžeme v logu generovaném `sender.exe` vidět, že uvedly jiný chybový kód než `0x5` nebo neindikovaly chybu vůbec, ale přesto nedošlo k jejich doručení. Přehled všech zpráv, které nebyly explicitně zablokovány UIPI filtrem, se nachází v tabulce A.9 a výsledky tohoto experimentu jsou rozebírány v sekci 4.4.4.

**Tabulka A.9** Jiné než blokované *window messages* (`PostMessage` s UIPI)

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| WM_NULL | 0x1 | 0x0 | doručená |
| WM_MOVE | 0x1 | 0x0 | doručená |
| WM_SIZE | 0x1 | 0x0 | doručená |
| WM_GETTEXT | 0x0 | 0x487 | pouze SendMessage |
| WM_GETTEXTLENGTH | 0x0 | 0x487 | pouze SendMessage |
| WM_GETHOTKEY | 0x1 | 0x0 | doručená |
| WM_GETICON | 0x1 | 0x0 | doručená |
| WM_GESTURE | 0x0 | 0x57 | nezprovozněná |
| 0x11b | 0x0 | 0x3ea | nezprovozněná |
| WM_DROPFILES | 0x0 | 0x6 | nezprovozněná |
| WM_TOUCH | 0x0 | 0x57 | nezprovozněná |
| WM_NCPOINTERUPDATE | 0x0 | 0x3ea | nezprovozněná |
| WM_NCPOINTERDOWN | 0x0 | 0x3ea | nezprovozněná |
| WM_NCPOINTERUP | 0x0 | 0x3ea | nezprovozněná |
| 0x244 | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERUPDATE | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERDOWN | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERUP | 0x0 | 0x3ea | nezprovozněná |
| 0x248 | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERENTER | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERLEAVE | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERACTIVATE | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERCAPTURECHANGED | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERWHEEL | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERHWHEEL | 0x0 | 0x3ea | nezprovozněná |
| DM_POINTERHITTEST | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERROUTEDTO | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERROUTEDAWAY | 0x0 | 0x3ea | nezprovozněná |
| WM_POINTERROUTEDRELEASED | 0x0 | 0x3ea | nezprovozněná |
| 0x254 | 0x0 | 0x3ea | nezprovozněná |
| 0x255 | 0x0 | 0x3ea | nezprovozněná |
| 0x256 | 0x0 | 0x3ea | nezprovozněná |

Tabulka pokračuje na další straně.

**Tabulka A.9** – pokračování z předchozí strany

| Zpráva | Návratová hodnota | Chybový kód | Stav |
|---|---|---|---|
| 0x257 | 0x0 | 0x3ea | nezprovozněná |
| 0x2cd | 0x0 | 0x0 | nezprovozněná |
| WM_RENDERFORMAT | 0x1 | 0x0 | doručená |
| WM_DRAWCLIPBOARD | 0x1 | 0x0 | doručená |
| WM_CHANGECBCHAIN | 0x1 | 0x0 | doručená |
| 0x313 | 0x1 | 0x0 | doručená |
| WM_THEMECHANGED | 0x1 | 0x0 | doručená |
| 0x31b | 0x1 | 0x0 | doručená |
| WM_DWMNCRENDERINGCHANGED | 0x1 | 0x0 | doručená |
| 0xc05f | 0x1 | 0x0 | doručená |
| 0xc060 | 0x1 | 0x0 | doručená |

### A.5.3 Manuální testování nedoručených zpráv

Některé *window messages* provádějí kontrolu `wParam` a `lParam` a vyžadují, aby obsahovaly konkrétní hodnoty či datové struktury. Testovací program `sender.exe` není dostatečně sofistikovaný na to, aby tyto požadavky během automatizovaného testování splnil, a proto jsou sporné a nedoručené zprávy dále manuálně testovány.

Pokud se jedná o oficiální zprávu, jsou její parametry upraveny dle dokumentace. Pro většinu zpráv byla postačující obecná úprava, při které je do `lParam` vložen ukazatel na libovolný objekt tak, jak je demonstrováno ve výpisu kódu A.3. V některých případech byly *window messages* požadovány speciální úpravy testovacího programu. Zdrojové kódy všech testovacích programů, včetně toho obecného, jsou dostupné na přiloženém médiu.

Pro zprávy, které nejsou zmíněny v oficiální dokumentaci, spočívalo manuální testování v pokusu vložení ukazatele na libovolný objekt nejprve do `lParam`, poté `wParam` a následně do obou parametrů.

### A.5.4 Další konfigurace *integrity levelů*

Jak již bylo řečeno v úvodu této sekce, pro aktivaci UIPI filtru postačí, aby byly aplikace `sender.exe` a `receiver.exe` spuštěny na různých *integrity levelech*. Chování filtru by mělo být na IL nezávislé, dokud `receiver.exe` běží s vyšším IL než `sender.exe`. Tento experiment toto tvrzení podkládá tím, že porovnává výstupy obou aplikací s výstupy analyzovanými v experimentech A.5.2.1 a A.5.2.2. Z důvodů vytyčených v experimentu A.3 však není možné při testování použít IL *untrusted*. Namátkou byly vybrány a srovnány následující konfigurace:

- funkce `SendMessage`, `sender.exe` s IL *low* a `receiver.exe` s IL *medium*;
- funkce `SendMessage`, `sender.exe` s IL *medium* a `receiver.exe` s IL *system*;
- funkce `PostMessage`, `sender.exe` s IL *low* a `receiver.exe` s IL *high*;
- funkce `PostMessage`, `sender.exe` s IL *high* a `receiver.exe` s IL *system*;
- funkce `PostMessage`, `sender.exe` s IL RID `0x2222` a `receiver.exe` s IL RID `0x3333`.

**Výpis kódu A.3** Manuální testování *window message* úpravou `lParam`

```
int main() {
    int dummy;

    std::cout << "SendMessage returned: 0x" << std::hex
              << SendMessage(HANDLE, WM_GETMINMAXINFO, 0,
                        reinterpret_cast<LPARAM>(&dummy)) << "\n"
              << "GetLastError returned: 0x" << GetLastError() << "\n";

    SetLastError(0);

    std::cout << "PostMessage returned: 0x" << std::hex
              << PostMessage(HANDLE, WM_GETMINMAXINFO, 0,
                        reinterpret_cast<LPARAM>(&dummy)) << "\n"
              << "GetLastError returned: 0x" << GetLastError() << "\n";

    return 0;
}
```

Můžeme pozorovat, že výstupy obou aplikaci se od výstupů použitých v experimentech A.5.2.1 a A.5.2.2 liší pouze ve dvojici známých zpráv patřících mezi řetězcové zprávy. Jak je rozebráno v sekci 4.4.6.8, tyto zprávy jsou registrovány CTF klientem a jejich identifikátory se mohou lišit. Za rozdíly nejsou považovány změny v *handlech* okna, časových značkách nebo pozicích kurzoru. Závěrem tedy je, že chování filtru UIPI je konzistentní napříč různými IL.

Stejným způsobem byly porovnány výstupy experimentů A.5.1.1 a A.5.1.2. Abychom se tentokrát vyvarovali přehlcení programu `receiver.exe`, jako tomu bylo v experimentu A.5.1.2, dočasně jej upravíme tak, aby nevypisoval žádné informace do konzole. Takto upravený program je na médiu dostupný pod jménem `receiver_no_print.exe`. Namátkou byly vybrány a srovnány následující konfigurace:

- funkce `SendMessage`, `sender.exe` s IL *low* a `receiver.exe` s IL *low*;
- funkce `PostMessage`, `sender.exe` s IL *high* a `receiver.exe` s IL *high*;
- funkce `SendMessage`, `sender.exe` s IL RID `0x2222` a `receiver.exe` s IL RID `0x2222`;
- funkce `PostMessage`, `sender.rexe` s IL *high* a `receiver.exe` s IL *low*.

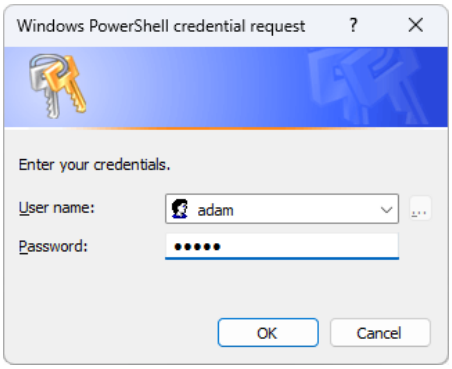
Můžeme pozorovat, že výstupy obou aplikací se od výstupů použitých v experimentech A.5.1.1 a A.5.1.2 liší v hodnotách parametrů některých doručených zpráv, různém počtu zpráv, které jsou doručeny v reakci na jiné zprávy, a různém načasování doručení zpráv, jež nebyly `sender.exe` odeslány. Dále můžeme narazit na rozdíly plynoucí z toho, že v dřívějších experimentech chyběly definice jmen pro zprávy `WM_POINTERDEVICECHANGE`, `WM_POINTERDEVICEINRANGE`, `WM_TABLET_ADDED` a další, které byly dodány až po jejich dokončení. Nicméně tyto rozdíly nemají vliv na to, které *window messages* jsou doručeny a které nikoliv, a můžeme tedy tento experiment uzavřít s tím, že chování aplikací bez aktivovaného UIPI filtru je konzistentní napříč *integrity levely*.

## A.6 Testování zpráv WM_GETTEXTLENGTH a WM_GETTEXT

Cílem tohoto experimentu je zjistit, jestli je možné zneužít *window messages* `WM_GETTEXTLENGTH` a `WM_GETTEXT` při aktivovaném UIPI filtru. Experiment probíhá následovně:

- Z programu `cmd.exe` běžícího s IL *high* spustíme prostřednictvím `launcher.bat` PowerShell skript `prompt_for_password.ps1`. Tím dojde k zobrazení dialogového okna žádajícího o zadání přihlašovacích údajů na obrázku A.3.
- Do políček *User name* a *Password* zadáme libovolné řetězce.
- Spustíme program `get_text.exe` s IL *medium*.

**Obrázek A.3** Dialogové okno žádající o zadání přihlašovacích údajů [158]




Můžeme pozorovat, že výstup programu zobrazený ve výpisu kódu A.4 indikuje úspěšné přečtení obsahu ovládacího prvku třídy *edit* obsahujícího uživatelské jméno. Můžeme si také povšimnout dalšího ovládacího prvku typu *edit*, jehož obsah je prázdný. Soudě podle sledu jednotlivých oken, případně vylučovací metodou, lze dojít k závěru, že tento ovládací prvek koresponduje s políčkem obsahujícím heslo. Z toho můžeme usoudit, že ovládací prvky určené pro práci s hesly implementují ochranu proti čtení pomocí zprávy `WM_GETTEXT`. Nicméně zpráva `WM_GETTEXTLENGTH` zůstává plně funkční a vrací správnou délku zadaného hesla. Implikace tohoto útoku jsou dále rozebírány v sekci 4.4.6.3.

**Výpis kódu A.4** Výstup programu `get_text.exe`

```
Found window of class: Edit
        Length: 24
        Contents: Enter your credentials.
Found window of class: SysCredential
        Length: 0
Found window of class: Static
        Length: 11
        Contents: &User name:
Found window of class: ComboBoxEx32
        Length: 4
        Contents: adam
Found window of class: ComboBox
        Length: 4
        Contents: ???
Found window of class: Edit
        Length: 4
        Contents: adam
Found window of class: Button
        Length: 4
        Contents: &...
Found window of class: Static
        Length: 10
        Contents: &Password:
Found window of class: Edit
        Length: 5
        Contents:
Found window of class: Button
        Length: 2
        Contents: OK
Found window of class: Button
        Length: 6
        Contents: Cancel
Found window of class: Static
        Length: 0
```

# Bibliografie


1. MICROSOFT CORPORATION. About Atom Tables. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-26]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/about-atom-tables.
2. DODSON, Donna; POLK, Tim; SOUPPAYA, Murugiah. *Securing Small-Business and Home Internet of Things (IoT) Devices.* NIST SP 1800-15A [online]. Gaithersburg (MA): National Institute of Standards a Technology, 2021 [cit. 2024-04-26]. Dostupné z DOI: https://doi.org/10.6028/NIST.SP.1800-15.
3. JANSEN, Wayne A.; WINOGRAD, Theodore; SCARFONE, Karen. *Guidelines on Active Content and Mobile Code.* NIST SP 800-28 Version 2 [online]. Gaithersburg (MA): National Institute of Standards a Technology, 2008 [cit. 2024-04-26]. Dostupné z DOI: https://doi.org/10.6028/NIST.SP.800-28ver2.
4. MOZILLA CORPORATION. Callback function. In: *MDN Web Docs* [online]. Mozilla Corporation, ©2024 [cit. 2024-04-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Callback_function.
5. KOKEŠ, Josef; ZAHRADNICKÝ, Tomáš. *Bezpečný kód: Útoky typu Denial-of-Service.* In: *Soubory Katerdy informační bezpečnosti* [online, prezentace]. Praha: ČVUT FIT, 2024 [cit. 2024-04-27]. Dostupné z: https://kib-files.fit.cvut.cz/bi-bek/recordings/2022/cz/prednaska_13.mp4.
6. MICROSOFT CORPORATION. Desktops. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-09]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winstation/desktops.
7. MICROSOFT CORPORATION. Dynamic-link library search order. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-26]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order.
8. JIRÁSEK, Petr; NOVÁK, Luděk; POŽÁR, Josef. *Výkladový slovník Kybernetické bezpečnosti.* 3. vyd. Praha: Policejní akademie ČR v Praze, 2015. ISBN 978-80-7251-397-0.
9. OWASP FOUNDATION. Testing for Local File Inclusion. In: *OWASP Web Security Testing Guide* [online]. OWASP Foundation, ©2024 [cit. 2024-04-26]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion.
10. MUZIKÁŘ, Zdeněk; ŽĎÁREK, Jan. *Administrace OS Unix: Virtualizace.* In: *OnlineFIT* [online]. Praha: ČVUT FIT, 2022 [cit. 2024-04-26]. Dostupné z: https://online.fit.cvut.cz/zaznam/B222/video/bi-adu.21_pre_2023-05-03.mp4.

11. KOKEŠ, Josef; ZAHRADNICKÝ, Tomáš. *Bezpečný kód*: *Bezpečnost webových aplikací*. In: *Soubory Katerdy informační bezpečnosti* [online]. Praha: ČVUT FIT, 2024 [cit. 2024-04-26]. Dostupné z: https://kib-files.fit.cvut.cz/bi-bek/recordings/2022/cz/prednaska_11.mp4.
12. CHESHIRE, Stuart; KROCHMAL, Marc. *Special-Use Domain Names.* NIST SP 800-28 Version 2 [online]. Cupertino (CA): Internet Engineering Task Force, 2013 [cit. 2024-04-26]. Dostupné z DOI: https://doi.org/10.17487/RFC6761.
13. MICROSOFT CORPORATION. LSA Logon Sessions. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-26]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthn/lsa-logon-sessions.
14. MICROSOFT CORPORATION. Using Messages and Message Queues. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-26]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/using-messages-and-message-queues.
15. IBM. What is debugging? In: *IBM Think* [online]. IBM, ©2024 [cit. 2024-04-26]. Dostupné z: https://www.ibm.com/topics/debugging.
16. TRACY, Miles; JANSEN, Wayne; SCARFONE, Karen; BUTTERFIELD, Jason. *Guidelines on Electronic Mail Security.* NIST SP 800-45 Version 2 [online]. Gaithersburg (MA): National Institute of Standards a Technology, 2007 [cit. 2024-04-26]. Dostupné z DOI: https://doi.org/10.6028/NIST.SP.800-45ver2.
17. OWASP FOUNDATION. Testing for Privilege Escalation. In: *OWASP Web Security Testing Guide* [online]. OWASP Foundation, ©2024 [cit. 2024-04-26]. Dostupné z: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/03-Testing_for_Privilege_Escalation.
18. DIOGENES, Yuri; OZKAYA, Erdal. *Cybersecurity – Attack and Defense Strategies: Infrastructure security with Red Team and Blue Team tactics*. 6th ed. Birmingham: Packt Publishing, 2018. ISBN 978-1-78847-529-7.
19. FRANKLIN, Joshua M.; HOWELL, Gema; BOECKL, Victoria Kaitlin; LEFKOVITZ, Naomi; NADEAU, Ellen. *Mobile Device Security.* NIST SP 1800-21 [online]. Gaithersburg (MA): National Institute of Standards a Technology, 2020 [cit. 2024-04-26]. Dostupné z DOI: https://doi.org/10.6028/NIST.SP.1800-21.
20. MUZIKÁŘ, Zdeněk; TRDLIČKA, Jan. *Unixové operační systémy*: *Úvod do operačního systému Unix*. In: *OnlineFIT* [online]. Praha: ČVUT FIT, 2021 [cit. 2024-04-27]. Dostupné z: https://courses.fit.cvut.cz/BI-UOS/lectures/bi-uos-p02-cli-01.pdf.
21. MICROSOFT CORPORATION. About Messages and Message Queues. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-26]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/about-messages-and-message-queues.
22. LAVERY, Oliver. *Win32 Message Vulnerabilities Redux: Shatter Attacks Remain a Threat* [online]. Reston (VA): IDEFENSE Inc., ©2003 [cit. 2024-03-03]. Tech. zpr. Dostupné z: https://web.archive.org/web/20030718164241/http:/www.idefense.com/idpapers/Shatter_Redux.pdf. Archivováno.
23. MICROSOFT CORPORATION. Window Procedures. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-26]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/window-procedures.
24. MICROSOFT CORPORATION. GetMessage function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getmessage.

25. MICROSOFT CORPORATION. Writing the Window Procedure. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/learnwin32/writing-the-window-procedure.
26. MICROSOFT CORPORATION. MSG structure (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/ns-winuser-msg.
27. KNOELLER, John. What's with „#ifdef _MAC" in Windows header files? In: *Stack Overflow* [online]. Stack Exchange, 2010 [cit. 2024-02-27]. Dostupné z: https://stackoverflow.com/questions/2376478/whats-with-ifdef-mac-in-windows-header-files.
28. MICROSOFT CORPORATION. WM_APP. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-app.
29. MICROSOFT CORPORATION. WM_USER. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-user.
30. ORMANDY, Tavis. CTFTOOL: Interactive CTF Exploration Tool. In: *GitHub* [online]. GitHub, ©2012 [cit. 2024-04-22]. Dostupné z: https://github.com/taviso/ctftool.
31. MICROSOFT CORPORATION. SendMessage function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage.
32. MICROSOFT CORPORATION. PostMessageA function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-postmessagea.
33. MICROSOFT CORPORATION. System Error Codes (0-499). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--0-499-.
34. MICROSOFT CORPORATION. WM_TIMER message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-timer.
35. BARBOSA, Edgar. *WINDOWS VISTA UIPI: User Interface Privilege Isolation* [online, prezentace]. [©2008]. [cit. 2024-03-03]. Dostupné z: https://web.archive.org/web/20120418173959/https://www.coseinc.com/en/index.php?rt=download&act=publication&file=Vista_UIPI.ppt.pdf. Archivováno.
36. MITRE CORPORATION. CWE-422: Unprotected Windows Messaging Channel („Shatter"). In: *Common Weakness Enumeration* [online]. MITRE Corporation, 2006 [cit. 2024-03-05]. Dostupné z: https://cwe.mitre.org/data/definitions/422.html.
37. PAGET, Chris. *Exploiting design flaws in the Win32 API for privilege escalation. Shatter Attacks – How to break Windows.* [online]. Tombom, 2002. [cit. 2024-03-03]. Tech. zpr. Dostupné z: https://web.archive.org/web/20060904080018/http://security.tombom.co.uk/shatter.html. Archivováno.
38. MICROSOFT CORPORATION. Microsoft Security Bulletin MS02-071 - Important. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-04]. Dostupné z: https://learn.microsoft.com/en-us/security-updates/SecurityBulletins/2002/ms02-071?redirectedfrom=MSDN.
39. MICROSOFT CORPORATION. Window Stations. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winstation/window-stations.

40. MICROSOFT CORPORATION. Information About Reported Architectural Flaw in Windows. In: *Microsoft TechNet* [online]. Microsoft Corporation, ©2012 [cit. 2024-03-04]. Dostupné z: https://web.archive.org/web/20121018185612/http://technet.microsoft.com/library/cc750553.aspx. Archivováno.
41. PAGET, Chris. *Shatter attacks - more techniques, more detail, more juicy goodness.* [online]. Tombom, 2002. [cit. 2024-03-04]. Tech. zpr. Dostupné z: https://web.archive.org/web/20060830211709/http://security.tombom.co.uk/moreshatter.html. Archivováno.
42. MICROSOFT CORPORATION. *Task Manager* [software]. 2001. [cit. 2024-04-09]. Dostupné z: https://www.microsoft.com/en-us/windows. Požadavky na systém: Windows NT 4.0 a vyšší.
43. NETWORK ASSOCIATES TECHNOLOGY. *McAfee Virus Scan 5.1* [software]. 2000. [cit. 2024-04-03]. Dostupné z: https://archive.org/details/mcafee-virusscan-5.1-cd. Požadavky na systém: nespecifikováno. Archivováno.
44. PAGET, Chris. *Shatter* [software]. 2002. [cit. 2024-04-01]. Dostupné z: https://web.archive.org/web/20060826200133/http://security.tombom.co.uk/shatter.zip. Požadavky na systém: nespecifikováno. Archivováno.
45. MICROSOFT CORPORATION. GetCursorPos function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getcursorpos.
46. MICROSOFT CORPORATION. WindowFromPoint function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-windowfrompoint.
47. MICROSOFT CORPORATION. EM_SETLIMITTEXT message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/em-setlimittext.
48. DARK SPYRIT. *jill.c* [software]. 2001. [cit. 2024-04-01]. Dostupné z: https://packetstormsecurity.com/files/24806/jill.c.html. Požadavky na systém: nespecifikováno.
49. MICROSOFT CORPORATION. WM_PASTE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-03]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-paste.
50. SIMONČIČ, Jernej. *netcat 1.12* [software]. [2011]. [cit. 2024-04-09]. Dostupné z: https://eternallybored.org/misc/netcat/. Požadavky na systém: Windows 32-bit nebo 64-bit.
51. MICROSOFT CORPORATION. *WinDbg* [software]. 2024. [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/. Požadavky na systém: operační systém Windows 11 nebo Windows 10 verze 1607 nebo vyšší, procesor x86 nebo ARM64.
52. PAGET, Chris. *Re: Security Vulnerability Report [MSRC 1250dg]* [elektronická pošta]. Zasláno z: secure@microsoft.com. 5. 8. 2002 14:24:06. [cit. 2024-03-03]. Dostupné z: https://web.archive.org/web/20060830033438/http://security.tombom.co.uk/response.txt. Osobní komunikace. Archivováno.
53. MICROSOFT CORPORATION. Interactive Services. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-04]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/services/interactive-services.

54. CLOSE, Tyler; CARP, Alan H.; STIEGLER, Marc. *Shatter-proofing Windows* [online]. Palo Alto (CA): Hewlett-Packard Laboratories, ©2004 [cit. 2024-03-04]. Tech. zpr. Dostupné z: https://www.blackhat.com/presentations/bh-usa-05/BH_US_05-Close/tylerclose_whitepaper_US05.pdf.
55. MICROSOFT CORPORATION. The Ten Immutable Laws of Security. In: *Microsoft TechNet* [online]. Microsoft Corporation, ©2002 [cit. 2024-03-05]. Dostupné z: https://web.archive.org/web/20021212220252/http://www.microsoft.com/technet/columns/security/essays/10imlaws.asp. Archivováno.
56. MOORE, Brett. *Shattering By Example* [online]. Security-Assessment.com, 2003. [cit. 2024-03-05]. Tech. zpr. Dostupné z: https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-moore/bh-us-04-moore-whitepaper.pdf.
57. MOORE, Brett. Shoot The Messenger: win32 Shatter Attacks. In: *Black Hat* [online]. Las Vegas (NV), 2004 [cit. 2024-03-05]. Dostupné z: https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-moore/bh-us-04-moore-up.ppt.
58. MICROSOFT CORPORATION. LVM_SORTITEMS message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/lvm-sortitems.
59. MICROSOFT CORPORATION. LVM_SORTITEMSEX message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/lvm-sortitemsex.
60. MICROSOFT CORPORATION. EM_SETWORDBREAKPROC message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/em-setwordbreakproc.
61. MICROSOFT CORPORATION. EM_SETWORDBREAKPROCEX message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/em-setwordbreakprocex.
62. MICROSOFT CORPORATION. EDITWORDBREAKPROCA callback function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nc-winuser-editwordbreakproca.
63. MICROSOFT CORPORATION. EDITWORDBREAKPROCEX callback function (richedit.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/richedit/nc-richedit-editwordbreakprocex.
64. MICROSOFT CORPORATION. EM_STREAMIN message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/em-streamin.
65. MICROSOFT CORPORATION. EM_STREAMOUT message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/em-streamout.
66. MICROSOFT CORPORATION. EDITSTREAM structure (richedit.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/richedit/ns-richedit-editstream.
67. MICROSOFT CORPORATION. EDITSTREAMCALLBACK callback function (richedit.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/richedit/nc-richedit-editstreamcallback.

68. MICROSOFT CORPORATION. EM_SETHYPHENATEINFO message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/controls/em-sethyphenateinfo`.
69. MICROSOFT CORPORATION. HYPHENATEINFO structure (richedit.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/api/richedit/ns-richedit-hyphenateinfo`.
70. MICROSOFT CORPORATION. HyphenateProc function (richedit.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/api/richedit/nf-richedit-hyphenateproc`.
71. MICROSOFT CORPORATION. TVM_SORTCHILDRENCB message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/controls/tvm-sortchildrencb`.
72. MICROSOFT CORPORATION. TVSORTCB structure (commctrl.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/api/commctrl/ns-commctrl-tvsortcb`.
73. MICROSOFT CORPORATION. TCM_GETITEMRECT message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/controls/tcm-getitemrect`.
74. MICROSOFT CORPORATION. RECT structure (windef.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/api/windef/ns-windef-rect`.
75. MICROSOFT CORPORATION. TCM_SETITEMSIZE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-05]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/controls/tcm-setitemsize`.
76. MITRE CORPORATION. CVE-2003-0659. In: *Common Vulnerabilities and Exposures* [online]. MITRE Corporation, ©1999–2024 [cit. 2024-03-09]. Dostupné z: `https://www.cve.org/CVERecord?id=CVE-2003-0659`.
77. MICROSOFT CORPORATION. CB_DIR message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-09]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/controls/cb-dir`.
78. MICROSOFT CORPORATION. LB_DIR message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-09]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/controls/lb-dir`.
79. MICROSOFT CORPORATION. Microsoft Security Bulletin MS03-045 - Important. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-09]. Dostupné z: `https://learn.microsoft.com/en-us/security-updates/securitybulletins/2003/ms03-045`.
80. MICROSOFT CORPORATION. Desktops. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-09]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/procthread/job-objects`.
81. CONOVER, Matthew. *Analysis of the Windows Vista Security Model* [online]. Symantec Corporation, [2006]. [cit. 2024-03-10]. Tech. zpr. Dostupné z: `https://web.archive.org/web/20120404064647/http://www.symantec.com/avcenter/reference/Windows_Vista_Security_Model_Analysis.pdf`. Archivováno.
82. RUSSINOVICH, Mark E.; SOLOMON, David A.; IONESCU, Alex. *Windows Internals: Part 1*. 6th ed. Redmond (WA): Microsoft Press, 2012. ISBN 978-0-7356-4873-9.

83. MICROSOFT CORPORATION. What is the Windows Integrity Mechanism? In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-10]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/bb625957(v=msdn.10).

84. BIBA, Kenneth J. *Integrity Considerations for Secure Computer Systems* [online]. Bedford (MA): The MITRE Corporation, 1975 [cit. 2024-03-10]. Tech. zpr. Dostupné z: https://seclab.cs.ucdavis.edu/projects/history/papers/biba75.pdf.

85. MICROSOFT CORPORATION. Security Descriptors. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-10]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptors.

86. MICROSOFT CORPORATION. Trustees. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-10]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/trustees.

87. MICROSOFT CORPORATION. Access Tokens. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-10]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/access-tokens.

88. MICROSOFT CORPORATION. Parts of the Access Control Model. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-18]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/access-control-components.

89. MICROSOFT CORPORATION. 2.1.4.2 Internal Components. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-10]. Dostupné z: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-azod/d28d536d-3973-4c8d-b2c9-989e3a8ba3c5.

90. MICROSOFT CORPORATION. SYSTEM_MANDATORY_LABEL_ACE structure (winnt.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-11]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-system_mandatory_label_ace.

91. MICROSOFT CORPORATION. Windows Integrity Mechanism Design. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-10]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/bb625963(v=msdn.10).

92. MICROSOFT CORPORATION. 2.4.2.4 Well-Known SID Structures. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-12]. Dostupné z: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/81d92bba-d22b-4a8c-908a-554ab29148ab.

93. MICROSOFT CORPORATION. ACE_HEADER structure (winnt.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-16]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-ace_header.

94. MICROSOFT CORPORATION. Mandatory integrity control in Windows Vista. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-11]. Dostupné z: https://learn.microsoft.com/en-us/archive/blogs/steriley/mandatory-integrity-control-in-windows-vista.

95. MICROSOFT CORPORATION. TOKEN_INFORMATION_CLASS enumeration (winnt.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-19]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ne-winnt-token_information_class.

96. MICROSOFT CORPORATION. TOKEN_MANDATORY_LABEL structure (winnt.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-19]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-token_mandatory_label.

97. MICROSOFT CORPORATION. Mandatory Integrity Control. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-12]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control.

98. MICROSOFT CORPORATION. Icacls. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-12]. Dostupné z: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/icacls.

99. MICROSOFT CORPORATION. AppContainer for legacy apps. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-12]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/appcontainer-for-legacy-applications-.

100. CHEN, Raymond. What are these SIDs of the form S-1-15-2-xxx? In: *The Old New Thing* [online]. Microsoft Corporation, 2022 [cit. 2024-03-12]. Dostupné z: https://devblogs.microsoft.com/oldnewthing/20220502-00/?p=106550.

101. RBMM. Windows process with Untrusted Integrity level. In: *Stack Overflow* [online]. Stack Exchange, 2017 [cit. 2024-03-12]. Dostupné z: https://stackoverflow.com/questions/44027935/windows-process-with-untrusted-integrity-level.

102. MICROSOFT CORPORATION. Designing Applications to Run at a Low Integrity Level. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-17]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/bb625960(v=msdn.10).

103. MICROSOFT CORPORATION. Can't preview Office documents in Outlook if Windows Firewall Service is disabled. In: *Microsoft Support* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-12]. Dostupné z: https://support.microsoft.com/en-gb/topic/7321f83c-ff40-3413-dbc9-97caf26d18d9.

104. POTTER, Jonathan. What does mandatory integrity level value of 0x2010 stand for? In: *Stack Overflow* [online]. Stack Exchange, 2015 [cit. 2024-03-12]. Dostupné z: https://stackoverflow.com/questions/31151139/what-does-mandatory-integrity-level-value-of-0x2010-stand-for.

105. JOHNSON, Jonny. Better know a data source: Process integrity levels. In: *Red Canary Blog* [online]. Red Canary, 2021 [cit. 2024-03-12]. Dostupné z: https://redcanary.com/blog/process-integrity-levels/.

106. MICROSOFT CORPORATION. *Protected Processes* [online]. Microsoft Corporation, 2006. [cit. 2024-03-12]. Tech. zpr. Dostupné z: https://download.microsoft.com/download/a/f/7/af7777e5-7dcd-4800-8a0a-b18336565f5b/process_vista.doc.

107. MICROSOFT CORPORATION. Protecting anti-malware services. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-12]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/services/protecting-anti-malware-services-.

108. MINASI, Mark. *Chml 1.53: change mandatory label* [software]. 2009. [cit. 2024-03-31]. Dostupné z: https://web.archive.org/web/20220528095357/https://www.minasi.com/apps/. Požadavky na systém: nespecifikováno. Archivováno.

109. MICROSOFT CORPORATION. Authorization Functions (Authorization). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-31]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/authorization-functions.

110. MICROSOFT CORPORATION. SACL Access Right. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-31]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/sacl-access-right.

111. MICROSOFT CORPORATION. Privilege Constants (Authorization). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-31]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/secauthz/privilege-constants.

112. RUSSINOVICH, Mark. *PsExec v2.43* [software]. 2023. [cit. 2024-03-31]. Dostupné z: https://learn.microsoft.com/en-us/sysinternals/downloads/psexec. Požadavky na systém: Windows 8.1 a vyšší, Windows Server 2012 a vyšší.

113. MICROSOFT CORPORATION. *Microsoft C/C++ Optimizing Compiler 19.38.33134* [software]. 2024. [cit. 2024-04-09]. Dostupné z: https://visualstudio.microsoft.com/downloads/. Požadavky na systém: procesor x86 nebo ARM64, operační systém Windows 10/11 nebo Windows Server 2016/2019/2022, operační paměť 4 GB, volné místo na disku minimálně 850 MB, grafická karta s rozlišením WXGA nebo lepším.

114. PYTHON SOFTWARE FOUNDATION. *Python 3.12.0:0fb18n0* [software]. 2023. [cit. 2024-04-09]. Dostupné z: https://www.python.org/downloads/release/python-3120/. Požadavky na systém: nespecifikováno.

115. MICROSOFT CORPORATION. Security descriptors in file systems. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/security-descriptors.

116. MICROSOFT CORPORATION. File System Functionality Comparison. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/fileio/filesystem-functionality-comparison.

117. MICROSOFT CORPORATION. 2.5.3.3 MandatoryIntegrityCheck Algorithm Pseudocode. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-18]. Dostupné z: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/ae69a089-473d-4c23-bf3d-7a12a9d11123.

118. MICROSOFT CORPORATION. SeAccessCheck function (wdm.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-18]. Dostupné z: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-seaccesscheck.

119. MICROSOFT CORPORATION. SECURITY_SUBJECT_CONTEXT structure (wdm.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-18]. Dostupné z: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_security_subject_context.

120. MICROSOFT CORPORATION. AccessCheck function (securitybaseapi.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-18]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/securitybaseapi/nf-securitybaseapi-accesscheck.

121. MICROSOFT CORPORATION. TOKEN_MANDATORY_POLICY structure (winnt.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-19]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-token_mandatory_policy.

122. MICROSOFT CORPORATION. 2.5.3.1.2 SidDominates. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-19]. Dostupné z: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/a4b0d7a9-bc25-43b9-af3a-6e0a1c17a205.

123. MICROSOFT CORPORATION. Modify an object label. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-03-24]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/security-policy-settings/modify-an-object-label.

124. MICROSOFT CORPORATION. The Windows Vista and Windows Server 2008 Developer Story: Windows Vista Application Development Requirements for User Account Control (UAC). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-02]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/aa905330(v=msdn.10).

125. MICROSOFT CORPORATION. SetWindowsHookExA function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa.

126. MICROSOFT CORPORATION. Application manifests. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-08]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/sbscs/application-manifests.

127. MICROSOFT CORPORATION. SendInput function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendinput.

128. MICROSOFT CORPORATION. ChangeWindowMessageFilter function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-changewindowmessagefilter.

129. MICROSOFT CORPORATION. ChangeWindowMessageFilterEx function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-changewindowmessagefilterex.

130. MICROSOFT CORPORATION. WM_CREATE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-create.

131. MICROSOFT CORPORATION. WM_NCCREATE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-nccreate.

132. MICROSOFT CORPORATION. WM_INITDIALOG message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dlgbox/wm-initdialog.

133. MICROSOFT CORPORATION. WM_DEVMODECHANGE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/gdi/wm-devmodechange.

134. MICROSOFT CORPORATION. WM_NOTIFY message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/wm-notify.

135. MICROSOFT CORPORATION. RegisterForTooltipDismissNotification function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-registerfortooltipdismissnotification.

136. MICROSOFT CORPORATION. DM_POINTERHITTEST message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-16]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/inputmsg/dm-pointerhittest.

137. MICROSOFT CORPORATION. WM_POINTERROUTEDTO message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-16]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/inputmsg/wm-pointerroutedto.

138. MICROSOFT CORPORATION. EM_GETLINE message (Winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-16]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/em-getline.

139. MICROSOFT CORPORATION. *Spy++ 14.00.25420* [software]. 2024. [cit. 2024-04-09]. Dostupné z: https://visualstudio.microsoft.com/downloads/. Požadavky na systém: procesor x86 nebo ARM64, operační systém Windows 10/11 nebo Windows Server 2016/2019/2022, operační paměť 4 GB, volné místo na disku minimálně 850 MB, grafická karta s rozlišením WXGA nebo lepším.

140. MICROSOFT CORPORATION. WM_MDICREATE message (Winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-16]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-mdicreate.

141. MICROSOFT CORPORATION. WM_DDE_TERMINATE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-terminate.

142. MICROSOFT CORPORATION. WM_DDE_ADVISE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-advise.

143. MICROSOFT CORPORATION. WM_DDE_UNADVISE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-unadvise.

144. MICROSOFT CORPORATION. WM_DDE_ACK message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-15]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-ack.

145. MICROSOFT CORPORATION. WM_DDE_DATA message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-data.

146. MICROSOFT CORPORATION. WM_DDE_REQUEST message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-request.

147. MICROSOFT CORPORATION. WM_DDE_POKE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-poke.

148. MICROSOFT CORPORATION. WM_DDE_EXECUTE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-dde-execute.

149. MICROSOFT CORPORATION. WM_DROPFILES message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-15]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/shell/wm-dropfiles.

150. MICROSOFT CORPORATION. System Error Codes (1000-1299). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-15]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--1000-1299-.

151. MICROSOFT CORPORATION. System Error Codes (1300-1699). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-15]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--1300-1699-.

152. MICROSOFT CORPORATION. WM_NULL message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-null.

153. MICROSOFT CORPORATION. WM_MOVE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-move.

154. MICROSOFT CORPORATION. WM_SIZE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-size.

155. CHEN, Raymond. Use WM_WINDOWPOSCHANGED to react to window state changes. In: *The Old New Thing* [online]. Microsoft Corporation, 2008 [cit. 2024-04-20]. Dostupné z: https://devblogs.microsoft.com/oldnewthing/20080115-00/?p=23813.

156. MICROSOFT CORPORATION. WM_GETTEXTLENGTH message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-gettextlength.

157. MICROSOFT CORPORATION. WM_GETTEXT message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-gettext.

158. MICROSOFT CORPORATION. *Windows PowerShell 5.1.22621.2506* [software]. 2016. [cit. 2024-04-22]. Dostupné z: https://www.microsoft.com/en-us/windows. Požadavky na systém: nespecifikováno.

159. MICROSOFT CORPORATION. Edit Control Text Operations. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-28]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/controls/edit-controls-text-operations.

160. MICROSOFT CORPORATION. Password Control Attribute. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-28]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/msi/password-control-attribute.

161. MICROSOFT CORPORATION. Control Attributes. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-28]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/msi/control-attributes.

162. MICROSOFT CORPORATION. WM_GETHOTKEY message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/inputdev/wm-gethotkey.

163. MICROSOFT CORPORATION. WM_GETICON message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-geticon.

164. MICROSOFT CORPORATION. WM_RENDERFORMAT message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-renderformat.

165. MICROSOFT CORPORATION. Clipboard Operations. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/clipboard-operations.

166. MICROSOFT CORPORATION. About the Clipboard. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/about-the-clipboard.

167. MICROSOFT CORPORATION. WM_RENDERALLFORMATS message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-20]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-renderallformats.

168. MICROSOFT CORPORATION. Using the Clipboard. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/using-the-clipboard.

169. MICROSOFT CORPORATION. WM_CHANGECBCHAIN message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-changecbchain.

170. ADEYBLUE. It Seems to be Some Sort of Communiqué. In: *Just Let It Flow* [online]. 2009 [cit. 2024-04-21]. Dostupné z: https://blog.airesoft.co.uk/2009/11/wm_messages/.

171. MICROSOFT CORPORATION. WM_THEMECHANGED message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/winmsg/wm-themechanged.

172. MICROSOFT CORPORATION. WM_DWMNCRENDERINGCHANGED message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dwm/wm-dwmncrenderingchanged.

173. MICROSOFT CORPORATION. Desktop Window Manager. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dwm/dwm-overview.

174. MICROSOFT CORPORATION. Custom Window Frame Using DWM. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/dwm/customframe.

175. MICROSOFT CORPORATION. RegisterWindowMessageA function. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-21]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-registerwindowmessagea.

176. HEXACORN. Enter Sandbox part 22: CTF – Capturing The … False (Positive Artifacts). In: *Hexacorn Tech Blog* [online]. Hexacorn, 2018 [cit. 2024-04-21]. Dostupné z: https://www.hexacorn.com/blog/2018/12/25/enter-sandbox-part-22-ctf-capturing-the-false-positive-artifacts/.

177. ORMANDY, Tavis. Down the Rabbit-Hole… In: *Project Zero* [online]. 2019 [cit. 2024-04-21]. Dostupné z: https://googleprojectzero.blogspot.com/2019/08/down-rabbit-hole.html.

178. MITRE CORPORATION. CVE-2019-1162. In: *Common Vulnerabilities and Exposures* [online]. MITRE Corporation, ©1999–2024 [cit. 2024-04-21]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-1162.

179. ORMANDY, Tavis. *ctftool 1.3* [software]. 2019. [cit. 2024-04-21]. Dostupné z: https://github.com/taviso/ctftool. Požadavky na systém: nespecifikováno.

180. MICROSOFT CORPORATION. Windows ALPC Elevation of Privilege Vulnerability. In: *Microsoft Security Response Center* [online]. Microsoft Corporation, 2019 [cit. 2024-04-21]. Dostupné z: https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2019-1162.

181. MICROSOFT CORPORATION. User Account Control: Switch to the secure desktop when prompting for elevation. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/security-policy-settings/user-account-control-switch-to-the-secure-desktop-when-prompting-for-elevation.

182. MICROSOFT CORPORATION. WM_MOUSEMOVE message. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/inputdev/wm-mousemove.

183. MICROSOFT CORPORATION. *Windows 11 Pro* [software]. 2021. [cit. 2024-04-09]. Dostupné z: https://www.microsoft.com/en-us/windows. Požadavky na systém: procesor x86 nebo ARM64 s výkonem 1 GHz nebo vyšším a dvěma jádry, operační paměť 4 GB, volné místo na 64 GB, UEFI, Secure Boot, TPM, grafická karta s kompatibilní s DirectX 12 nebo novějším a driverem WDDM 2.0.

184. MICROSOFT CORPORATION. *Microsoft Incremental Linker 19.38.33134* [software]. 2024. [cit. 2024-04-09]. Dostupné z: https://visualstudio.microsoft.com/downloads/. Požadavky na systém: procesor x86 nebo ARM64, operační systém Windows 10/11 nebo Windows Server 2016/2019/2022, operační paměť 4 GB, volné místo na disku minimálně 850 MB, grafická karta s rozlišením WXGA nebo lepším.

185. RUSSINOVICH, Mark. *Process Explorer v17.05* [software]. 2023. [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer. Požadavky na systém: Windows 8.1 a vyšší, Windows Server 2012 a vyšší.

186. MICROSOFT CORPORATION. *GFlags 10.0.22621.0* [software]. 2024. [cit. 2024-04-09]. Dostupné z: https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/. Požadavky na systém: procesor s výkonem 1.6 GHz nebo vyšším, operační systém Windows 10 verze 1507 nebo vyšší nebo Windows Server 2012/2016/2019/2022 nebo Windows 8.1 nebo Windows 7 SP1, operační paměť 1 GB, volné místo na disku 4 GB.

187. MICROSOFT CORPORATION. NTFS overview. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview.

188. MICROSOFT CORPORATION. FAT32 File System Specification. In: *Windows Hardware Developer Central* [online]. Microsoft Corporation, 2000 [cit. 2024-04-22]. Dostupné z: https://web.archive.org/web/20070104012128/http://microsoft.com/whdc/system/platform/firmware/fatgen.mspx. Archivováno.

189. MICROSOFT CORPORATION. exFAT file system specification. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-22]. Dostupné z: https://learn.microsoft.com/en-US/windows/win32/fileio/exfat-specification.

190. MICROSOFT CORPORATION. Show loader snaps. In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/show-loader-snaps.

191. MICROSOFT CORPORATION. MonitorFromWindow function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-monitorfromwindow.

192. MICROSOFT CORPORATION. MonitorFromPoint function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-09]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-monitorfrompoint.

193. MICROSOFT CORPORATION. IsWindow function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-iswindow?redirectedfrom=MSDN).

194. MICROSOFT CORPORATION. IsWindowEnabled function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-iswindowenabled.

195. MICROSOFT CORPORATION. IsWindowVisible function (winuser.h). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-06]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-iswindowvisible`.

196. MICROSOFT CORPORATION. System Error Codes (1700-3999). In: *Microsoft Learn* [online]. Microsoft Corporation, ©2024 [cit. 2024-04-15]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes--1700-3999-`.

# Obsah příloh

- `readme.md` — stručný popis obsahu média
- `experiments` — výstupy experimentů a menší testovací programy
  - `A-4` — experiment „Kontrola platnosti *handle* okna s vyšším *integrity levelem*“
    - `build` — spustitelná forma implementace
    - `source` — zdrojový kód implementace
  - `A-5-1-1` — experiment „Chování filtru *window messages*“ – `SendMessage` bez UIPI
    - `receiver_WindowProc.csv` — výstup `WindowProc` programu `receiver.exe`
    - `receiver_wWinMain.csv` — výstup `wWinMain` programu `receiver.exe`
    - `sender.csv` — výstup programu `sender.exe`
  - `A-5-1-2` — experiment „Chování filtru *window messages*“ – `PostMessage` bez UIPI
    - … — struktura popsána v `readme.md` v adresáři
  - `A-5-2-1` — experiment „Chování filtru *window messages*“ – `SendMessage` s UIPI
    - … — stejná vnitřní struktura jako u A-5-1-1
  - `A-5-2-2` — experiment „Chování filtru *window messages*“ – `PostMessage` s UIPI
    - … — stejná vnitřní struktura jako u A-5-1-1
  - `A-5-3` — experiment „Chování filtru *window messages*“ – manuální testování
    - … — stejná vnitřní struktura jako u A-4
  - `A-5-4` — experiment „Chování filtru *window messages*“ – další konfigurace *integrity levelů*
    - … — struktura popsána v `readme.md` v adresáři
  - `A-6` — experiment „Testování zpráv `WM_GETTEXTLENGTH` a `WM_TEXTLENGTH`“
    - … — stejná vnitřní struktura jako u A-4
- `il_file` — program pro nastavení IL na souboru či adresáři
  - … — stejná vnitřní struktura jako u A-4
- `il_proc` — program pro nastavení IL na procesu
  - … — stejná vnitřní struktura jako u A-4
- `sender-receiver` — programy pro testování filtru *window messages*
  - … — stejná vnitřní struktura jako u A-4
- `thesis` — bakalářská práce
  - `build` — text práce ve formátu *Portable Document Format* (PDF)
  - `source` — zdrojový kód ve formátu LaTeX