# Bezpieczeństwo Windows od środka

*Kompleksowe spojrzenie na uwierzytelnianie, autoryzację i audyt systemu*

James Forshaw

Helion

no starch press

Tytuł oryginału: Windows Security Internals: A Deep Dive into Windows Authentication, Authorization, and Auditing

Tłumaczenie: Piotr Pilch

ISBN: 978-83-289-2065-1



Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*helion.pl/user/opinie/bezwin*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

# Spis treści

## Część II
## MONITOR SRM

## CZĘŚĆ III
## UWIERZYTELNIANIE I AUTORYZACJA W RAMACH ZABEZPIECZEŃ LOKALNYCH

# 4

# Tokeny dostępu bezpieczeństwa

*Token dostępu bezpieczeństwa* (ang. *security access token*) lub w skrócie *token* to „serce” zabezpieczeń systemu Windows. Monitor SRM (*Security Reference Monitor*) używa najpierw tokenów do reprezentowania tożsamości, takich jak konta użytkowników, a następnie udziela im dostępu do zasobów lub go odmawia. W systemie Windows tokeny są reprezentowane za pomocą obiektów `Token` jądra, zawierających jako minimum określoną, reprezentowaną przez siebie tożsamość, wszelkie obejmujące ją grupy zabezpieczeń oraz specjalne uprawnienia nadane tożsamości.

Podobnie jak inne obiekty jądra, tokeny obsługują informacyjne wywołania systemowe `Query` i `Set`, które pozwalają użytkownikowi na sprawdzanie właściwości tokena oraz ustawianie określonych właściwości. Choć rzadziej używane, niektóre interfejsy API podsystemu Win32, takie jak na przykład `GetTokenInformation` i `SetTokenInformation`, również udostępniają te dwa wywołania systemowe.

Zacznijmy od przeglądu dwóch głównych typów tokenów, z którymi możesz się spotkać podczas analizy zabezpieczeń systemu Windows. Są to tokeny podstawowe i tokeny personifikacji. Dalej szczegółowo zostanie omówionych wiele istotnych właściwości, które token zawiera. Musisz je zrozumieć, zanim przejdziesz do omówienia kontroli dostępu w rozdziale 7.

# Tokeny podstawowe

Każdy proces ma przypisany token, który opisuje jego tożsamość dla dowolnej operacji dostępu do zasobów. Gdy monitor SRM przeprowadza kontrolę dostępu, kieruje zapytanie do tokena procesu i używa go do określenia, jaki rodzaj dostępu ma przyznać. W przypadku zastosowania tokena na potrzeby procesu jest on nazywany *tokenem podstawowym* (ang. *primary token*).

Token procesu możesz otworzyć za pomocą wywołania systemowego `NtOpenProcessToken` zwracającego uchwyt, za pomocą którego można pobrać informacje o tokenie. Obiekt `Token` to zasób umożliwiający zabezpieczenie, więc aby uzyskać uchwyt, obiekt wywołujący musi przejść kontrolę dostępu. Zauważ, że aby mieć możliwość uzyskania informacji o tokenie, konieczny jest również uchwyt procesu z prawem dostępu `QueryLimitedInformation`.

Podczas otwierania obiektu `Token` można zażądać następujących praw dostępu:

**`AssignPrimary`**. Przypisuje obiekt `Token` jako token podstawowy.

**`Duplicate`**. Duplikuje obiekt `Token`.

**`Impersonate`**. Dokonuje personifikacji obiektu `Token`.

**`Query`**. Uzyskuje właściwości obiektu `Token`, takie jak jego grupy i uprawnienia.

**`QuerySource`**. Uzyskuje źródło obiektu `Token`.

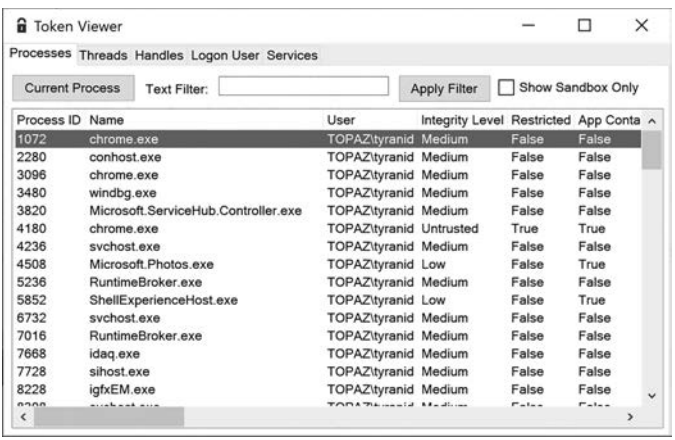
**`AdjustPrivileges`**. Dostosowuje listę uprawnień obiektu `Token`.

**`AdjustGroups`**. Dostosowuje listę grup obiektu `Token`.

**`AdjustDefault`**. Dostosowuje właściwości obiektu `Token` nieobjęte przez inne prawa dostępu.

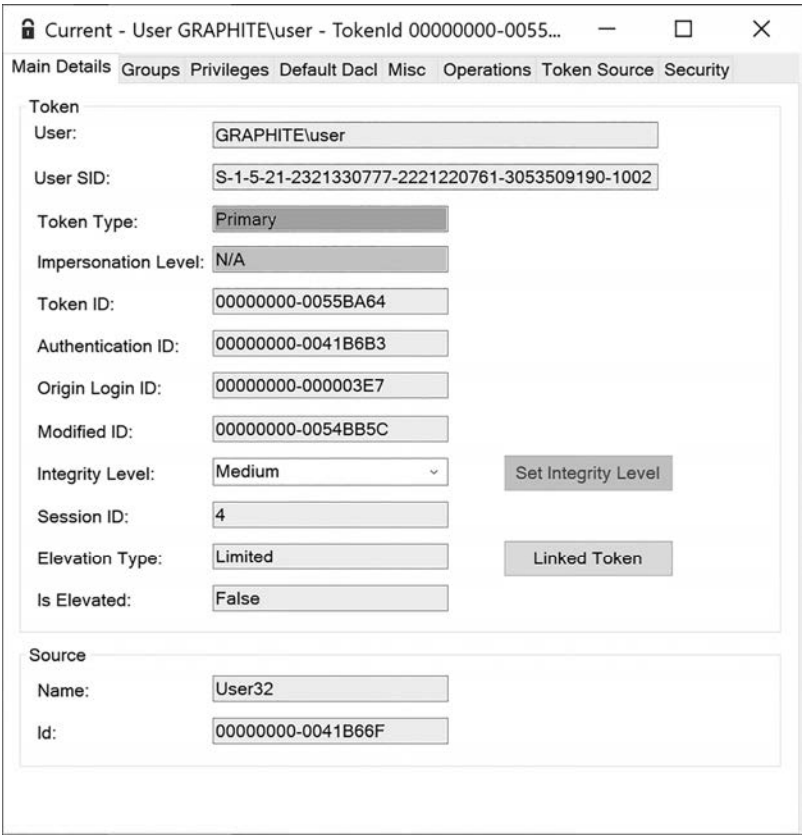
**`AdjustSessionId`**. Dostosowuje identyfikator sesji obiektu `Token`.

Listę dostępnych procesów i ich tokenów możesz zobaczyć po uruchomieniu polecenia `Show-NtToken –All` narzędzia PowerShell. Powinno ono spowodować otworzenie aplikacji Token Viewer (rysunek 4.1).




*Rysunek 4.1. Aplikacja Token Viewer zawiera listę wszystkich dostępnych procesów i ich tokenów*

Widok listy oferuje jedynie prosty przegląd dostępnych tokenów. Aby uzyskać więcej informacji, kliknij dwukrotnie jedną z pozycji procesów, co spowoduje wyświetlenie szczegółowego widoku tokena (rysunek 4.2).




*Rysunek 4.2. Szczegółowy widok obiektu Token procesu*

Zwróćmy uwagę na kilka istotnych porcji informacji w tym widoku. U samej góry znajduje się nazwa użytkownika i jego identyfikator SID. Obiekt `Token` przechowuje jedynie identyfikator, ale widok tokena udostępni nazwę, jeśli jest dostępna. Następne pole wskazuje typ tokena. Jako że sprawdzany jest token podstawowy, ustawiono typ `Primary`. Poziom personifikacji (poniżej tego pola) używany jest wyłącznie dla tokenów personifikacji, które zostaną omówione w dalszej części rozdziału. Poziom ten nie jest wymagany w przypadku tokenów podstawowych, dlatego ustawiono wartość *N/A*.

W środkowej części okna dialogowego znajduje się lista następujących czterech 64-bitowych identyfikatorów całkowitoliczbowych:

***Token ID*** (identyfikator tokena). Unikalna wartość przypisana w momencie utworzenia obiektu `Token`.

***Authentication ID*** (identyfikator uwierzytelniania). Wartość wskazująca sesję logowania, do której należy token.

***Origin Login ID*** (identyfikator pierwotnego logowania). Identyfikator uwierzytelnienia nadrzędnej sesji logowania.

***Modified ID*** (identyfikator modyfikacji). Unikalna wartość aktualizowana w momencie zmodyfikowania określonych wartości tokena.

Gdy użytkownik uwierzytelnia się na komputerze z systemem Windows, podsystem LSASS tworzy *sesję logowania*. Śledzi ona zasoby związane z uwierzytelnieniem użytkownika. Sesja przechowuje na przykład kopię danych uwierzytelniających użytkownika, aby mogły one zostać ponownie wykorzystane. Podczas procesu tworzenia sesji logowania monitor SRM generuje unikalny identyfikator uwierzytelnienia, który może zostać użyty do odwoływania się do sesji. W związku z tym w przypadku danej sesji logowania wszystkie tokeny użytkownika będą mieć ten sam identyfikator uwierzytelnienia. Jeśli użytkownik uwierzytelnia się dwukrotnie na tym samym komputerze, monitor SRM generuje różne identyfikatory uwierzytelnienia.

Identyfikator pierwotnego logowania wskazuje, kto utworzył sesję logowania tokena. Jeżeli uwierzytelnisz innego użytkownika na swoim pulpicie (wywołując na przykład interfejs API `LogonUser` z nazwą użytkownika i hasłem), identyfikator pierwotnego logowania będzie pełnił rolę identyfikatora uwierzytelnienia tokena obiektu wywołującego. Zauważ, że pole to widoczne na rysunku 4.2 zawiera wartość *00000000-000003E7*. Jest to jeden z czterech ustalonych identyfikatorów uwierzytelnienia definiowanych przez monitor SRM, wskazujący w tym przypadku na sesję logowania *SYSTEM*. W tabeli 4.1 zamieszczono cztery ustalone wartości wraz z identyfikatorami SID kont użytkowników powiązanych z sesjami.

*Tabela 4.1. Identyfikatory uwierzytelnienia i identyfikatory SID użytkowników w przypadku ustalonych sesji logowania*

| Identyfikator uwierzytelnienia | Identyfikator SID użytkownika | Nazwa użytkownika sesji logowania |
|---|---|---|
| 00000000-000003E4 | S-1-5-20 | *NT AUTHORITY\NETWORK SERVICE* |
| 00000000-000003E5 | S-1-5-19 | *NT AUTHORITY\LOCAL SERVICE* |
| 00000000-000003E6 | S-1-5-7 | *NT AUTHORITY\ANONYMOUS LOGON* |
| 00000000-000003E7 | S-1-5-18 | *NT AUTHORITY\SYSTEM* |

Poniżej identyfikatorów w widoku szczegółowym znajduje się pole wskazujące *poziom integralności* tokena. Poziom integralności po raz pierwszy wprowadzono w systemie Windows Vista w celu implementacji prostego mechanizmu *obowiązkowej kontroli dostępu*, w ramach którego dostęp do zasobów wymuszają zasady systemowe, zatem nie pozwala się na to, by poszczególne zasoby określały własne zasady dostępu. Poziomy integralności zostaną omówione w podrozdziale „Grupy tokenów” tego rozdziału.

Dalej pojawia się identyfikator sesji, czyli numer przypisany sesji konsoli, z którą jest powiązany proces. Nawet pomimo tego, że sesja konsoli jest właściwością procesu, jej wartość jest określona w tokenie procesu.

### LOKALNIE UNIKALNE IDENTYFIKATORY

Wspomniałem, że identyfikatory tokena to 64-bitowe liczby całkowite. Technicznie rzecz biorąc, są to struktury *lokalnie unikalnych identyfikatorów* (LUID — *Locally Unique Identifier*), które zawierają dwie 32-bitowe wartości. Identyfikatory LUID to powszechny typ w systemie, a monitor SRM używa ich, gdy wymagana jest unikalna wartość. Są one na przykład stosowane do identyfikowania w sposób unikalny wartości uprawnień.

Używając wywołania systemowego `NtAllocateLocallyUniqueId` lub polecenia `Get-NtLocallyUniqueId` narzędzia PowerShell, możesz przydzielić własny identyfikator LUID. Gdy skorzystasz z wywołania systemowego, system Windows upewni się, czy dysponuje centralnym elementem autoryzacji pozwalającym wygenerować kolejny unikalny identyfikator. Jest to ważne, ponieważ ponowne użycie wartości mogłoby mieć katastrofalne skutki. Jeśli na przykład identyfikator LUID zostałby ponownie zastosowany jako identyfikator uwierzytelnienia tokena, mógłby się pokryć z jednym z identyfikatorów zdefiniowanych w tabeli 4.1. Mogłoby to wprowadzić system w błąd, bo „pomyślałby", że bardziej uprzywilejowany użytkownik uzyskiwał dostęp do zasobu, co skutkowałoby zwiększeniem uprawnień.

Graficzny interfejs użytkownika aplikacji Token Viewer okaże się znakomity, gdy zechcesz ręcznie sprawdzić informacje o tokenie. Aby zapewnić sobie dostęp w sposób programowy, możesz otworzyć obiekt `Token` w narzędziu PowerShell za pomocą polecenia `Get-NtToken`. W celu uzyskania tokena bieżącego procesu wykonaj następujące polecenie:

```
PS> $token = Get-NtToken
```

Jeśli chcesz otworzyć token konkretnego procesu, możesz użyć następującego polecenia, zastępując łańcuch *<PID>* identyfikatorem procesu docelowego:

```
PS> $token = Get-NtToken -ProcessId <PID>
```

Wynikiem polecenia `Get-NtToken` jest obiekt `Token`, którego właściwości możesz uzyskać. Możesz na przykład wyświetlić użytkownika tokena, co prezentuje listing 4.1.

*Listing 4.1. Wyświetlenie użytkownika za pomocą właściwości obiektu Token*

```
PS> $token.User
Name              Attributes
----              ----------
GRAPHITE\user     None
```

Aby wyświetlić podstawowe informacje w konsoli, zastosuj polecenie `Format-NtToken` (listing 4.2).

*Listing 4.2. Wyświetlenie właściwości tokena za pomocą polecenia Format-NtToken*

```
PS> Format-NtToken $token -All
USER INFORMATION
----------------
Name                  Attributes
----                  ----------
GRAPHITE\user         None

GROUP SID INFORMATION
---------------------
Name                  Attributes
----                  ----------
GRAPHITE\None         Mandatory, EnabledByDefault
Everyone              Mandatory, EnabledByDefault
--cięcie--
```

Masz możliwość przekazania poleceniu `Show-NtToken` otwartego obiektu `Token`, co spowoduje wyświetlenie tego samego interfejsu GUI, który pokazano na rysunku 4.2.

# Tokeny personifikacji

*Token personifikacji* (ang. *impersonation token*) to drugi typ tokena, z którym się spotkasz. Tokeny te są szczególnie istotne w przypadku usług systemowych, ponieważ pozwalają procesowi o jednej tożsamości tymczasowo podszywać się pod inną tożsamość w celu przeprowadzenia kontroli dostępu. Na przykład usługa może wymagać podczas wykonywania pewnej operacji otwarcia pliku należącego do innego użytkownika. Zezwalając tej usłudze na podszycie się pod użytkownika wywołującego, system przyznaje jej dostęp do pliku, nawet jeśli usługa nie mogłaby bezpośrednio otworzyć pliku.

Tokeny personifikacji są przypisywane wątkom, a nie procesom. Oznacza to, że tylko kod uruchomiony w danym wątku przyjmie tożsamość, pod którą się podszyto. Istnieją następujące trzy sposoby przypisywania wątkowi tokena personifikacji:

- jawne nadanie obiektowi `Token` prawa dostępu `Impersonate` oraz obiektowi `Thread` prawa dostępu `SetThreadToken`,
- jawne nadanie obiektowi `Thread` prawa dostępu `DirectImpersonation`,
- niejawnie drogą personifikacji żądanie wywołania RPC.

Najprawdopodobniej spotkasz się z niejawnym przypisaniem tokena, ponieważ jest to najczęstszy wariant w przypadku usług systemowych, które udostępniają mechanizmy wywołań RPC. Jeśli na przykład usługa tworzy serwer nazwanego potoku, może podszywać się pod klientów, którzy łączą się z potokiem za pomocą interfejsu API `ImpersonateNamedPipe`. Gdy utworzono wywołanie względem nazwanego potoku, jądro przechwytuje *kontekst personifikacji* na podstawie wątku wywołującego i procesu. Kontekst ten służy do

przypisania tokena personifikacji wątkowi, który wywołuje interfejs `ImpersonateNamedPipe`. Kontekst personifikacji może bazować na istniejącym tokenie personifikacji wątku lub na kopii tokena podstawowego procesu.

## Opcja SQoS

Co, jeśli nie chcesz, aby usługa mogła podszywać się pod Twoją tożsamość? Monitor SRM obsługuje opcję *SQoS* (*Security Quality of Service*), która umożliwia kontrolowanie tego. Po otwarciu nazwanego potoku za pomocą interfejsów API systemu plików możesz przekazać strukturę `SECURITY_QUALITY_OF_SERVICE` w polu `SecurityQualityOfService` struktury `OBJECT_ATTRIBUTES`. Struktura opcji SQoS zawiera trzy wartości konfiguracyjne: poziom personifikacji, tryb śledzenia kontekstu oraz tryb efektywnego tokena.

Najważniejszym polem opcji SQoS kontrolującym, co usługa może zrobić z Twoją tożsamością, jest *poziom personifikacji*. Definiuje on poziom dostępu przyznany usłudze podczas niejawnego personifikowania użytkownika wywołującego. Poziom może być reprezentowany przez jedną z następujących czterech wartości wymienionych zgodnie z rosnącym stopniem uprawnień:

1. **`Anonymous` (anonimowość)**. Uniemożliwia usłudze otwarcie obiektu `Token` i sprawdzenie tożsamości użytkownika. Jest to najniższy poziom. Jeśli obiekt wywołujący ustawił ten poziom, działać będzie tylko ograniczony zestaw usług.
2. **`Identification` (identyfikacja)**. Pozwala usłudze otworzyć obiekt `Token` i uzyskać tożsamość, grupy i uprawnienia użytkownika. Podczas podszywania się pod użytkownika wątek nie może jednak otworzyć żadnych chronionych zasobów.
3. **`Impersonation` (personifikacja)**. Umożliwia usłudze skorzystanie w pełni z tożsamości użytkownika w systemie lokalnym. Usługa może otwierać zasoby lokalne chronione przez użytkownika i modyfikować je. Może ona również uzyskiwać dostęp do zasobów zdalnych, jeśli użytkownik jest lokalnie uwierzytelniony w systemie. Jeżeli jednak użytkownik uwierzytelnił się za pośrednictwem połączenia sieciowego, takiego jak oparte na protokole SMB (*Server Message Block*), usługa nie może użyć obiektu `Token` do uzyskania dostępu do zasobów zdalnych.
4. **`Delegation` (delegowanie)**. Pozwala usłudze otworzyć wszystkie zasoby lokalne i zdalne, tak jakby była użytkownikiem. Jest to najwyższy poziom. Aby jednak uzyskać dostęp do zasobu zdalnego w przypadku użytkowników uwierzytelnionych z użyciem sieci, nie wystarczy dysponować tym poziomem personifikacji. W celu umożliwienia tego musi też zostać skonfigurowana domena systemu Windows. Ten poziom personifikacji zostanie dokładniej omówiony w rozdziale 14. dotyczącym uwierzytelniania z użyciem protokołu Kerberos.

Poziom personifikacji możesz określić w ramach opcji SQoS albo przy wywoływaniu usługi lub podczas tworzenia kopii istniejącego tokena. Aby ograniczyć, na co usługa pozwala, ustaw poziom identyfikacji lub anonimowości. Zapobiegnie to uzyskaniu przez usługę dostępu do jakichkolwiek zasobów, choć na poziomie identyfikacji nadal będzie mogła ona używać tokena i wykonywać operacje w imieniu obiektu wywołującego.

Wykonajmy test za pomocą polecenia `Invoke-NtToken` narzędzia PowerShell. W listingu 4.3 dokonano personifikacji tokena na dwóch różnych poziomach i podjęto próbę uruchomienia skryptu, który otwiera zabezpieczony zasób. Poziom personifikacji określono przy użyciu właściwości `ImpersonationLevel`.

*Listing 4.3. Personifikowanie tokena na różnych poziomach i otwieranie zabezpieczonego zasobu*

```
PS> $token = Get-NtToken
PS> Invoke-NtToken $token {
    Get-NtDirectory -Path "\"
} -ImpersonationLevel Impersonation
Name NtTypeName
---- ----------
     Directory

PS> Invoke-NtToken $token {
    Get-NtDirectory -Path "\"
} -ImpersonationLevel Identification
Get-NtDirectory : (0xC00000A5) - A specified impersonation level is invalid.
--cięcie--
```

W ramach pierwszego wykonanego polecenia uzyskano uchwyt tokena podstawowego bieżącego procesu. Dalej wywołano polecenie `Invoke-NtToken`, aby dokonać personifikacji tokena na poziomie `Impersonation` i uruchomić skrypt, który uruchamia polecenie `Get-NtDirectory` w celu otwarcia katalogu głównego przestrzeni OMNS. Operacja otwarcia udaje się, a obiekt katalogu wyświetlany jest w konsoli.

Spróbowano następnie powtórzyć operację na poziomie identyfikacji, ale tym razem otrzymano błąd `STATUS_BAD_IMPERSONATION_LEVEL` (jeśli napotkasz ten błąd podczas tworzenia aplikacji lub korzystania z systemu, będziesz już znać jego przyczynę!). Zauważ, że operacja otwierania nie zwraca błędu odmowy dostępu, ponieważ monitor SRM nie osiąga aż tak odległego etapu, aby sprawdzić, czy podszywający się użytkownik może uzyskać dostęp do zasobu.

### UŻYTKOWNICY ANONIMOWI

Określenie poziomu personifikacji `Anonymous` nie jest tym samym co działanie z uprawnieniami użytkownika *ANONYMOUS LOGON*, do którego odwołano się w tabeli 4.1. Możliwe jest realizowanie działań z użyciem tożsamości użytkownika anonimowego i uzyskanie dostępu do zasobu drogą kontroli dostępu, ale token na poziomie anonimowym nie może przejść żadnej takiej kontroli niezależnie od tego, jak skonfigurowano zabezpieczenia zasobu.

Jądro implementuje wywołanie systemowe `NtImpersonateAnonymousToken`, które pozwala podszyć się pod użytkownika anonimowego w przypadku określonego wątku. Możliwe jest również uzyskanie dostępu do tokena użytkownika anonimowego za pomocą polecenia `Get-NtToken`:

```
PS> Get-NtToken -Anonymous | Format-NtToken
NT AUTHORITY\ANONYMOUS LOGON
```

Dwa pozostałe pola w strukturze opcji SQoS są rzadziej używane, ale nadal istotne. *Tryb śledzenia kontekstu* decyduje o tym, czy tożsamość użytkownika ma zostać statycznie przechwycona w momencie nawiązywania połączenia z usługą. Jeżeli tożsamość nie zostanie w ten sposób przechwycona, a obiekt wywołujący podszyje się następnie pod innego użytkownika przed wywołaniem usługi, nowa personifikowana tożsamość stanie się dostępna dla usługi, a nie dla procesu. Zauważ, że taka tożsamość może zostać przekazana usłudze tylko wtedy, gdy jest na poziomie `Impersonation` lub `Delegation`. Jeśli token poddany personifikacji jest na poziomie identyfikacji lub anonimowym, monitor SRM wygeneruje błąd zabezpieczeń i odrzuci operację podszywania się.

*Tryb efektywnego tokena* modyfikuje w inny sposób token przekazywany serwerowi. Możliwe jest wyłączenie grup i uprawnień przed utworzeniem połączenia, a jeśli tryb ten wyłączono, serwer może ponownie je włączyć i z nich korzystać. Jeżeli jednak tryb efektywnego tokena jest włączony, monitor SRM usunie te grupy i uprawnienia, aby serwer nie mógł ich ponownie aktywować ani zastosować.

Domyślnie, jeśli podczas otwierania kanału komunikacji międzyprocesowej (IPC — *Interprocess Communication*) nie zostanie określona żadna struktura opcji SQoS, poziom obiektu wywołującego to poziom `Impersonation` ze statycznym śledzeniem i nieefektywnym tokenem. Jeżeli przechwycono kontekst personifikacji, a obiekt wywołujący już się podszywa, poziom personifikacji tokena wątku musi być równy poziomowi `Impersonation` lub wyższy. W przeciwnym razie przechwycenie zakończy się niepowodzeniem. Jest to ważna opcja zabezpieczeń, która zapobiega utworzeniu przez obiekt wywołujący na poziomie identyfikacji lub niższym wywołania za pośrednictwem kanału RPC i udawaniu innego użytkownika.

**Uwaga**

*Opisałem sposób określania opcji SQoS na poziomie natywnych wywołań systemowych, ponieważ struktura `SECURITY_QUALITY_OF_SERVICE` nie jest bezpośrednio udostępniana przez interfejsy API podsystemu Win32. Zazwyczaj jest ona określana za pomocą dodatkowych flag. Na przykład interfejs `CreateFile` udostępnia opcję SQoS przez ustawienie flagi `SECURITY_SQOS_PRESENT`.*

## Jawna personifikacja tokena

Istnieją dwa sposoby jawnej personifikacji tokena (podszywania się pod niego). Jeśli dysponujesz uchwytem personifikacji obiektu `Token` z prawem dostępu `Impersonate`, możesz przypisać go do wątku za pomocą wywołania systemowego `NtSetInformationThread` i klasy informacji `ThreadImpersonationToken`.

Gdy zamiarem jest podszywanie się pod dany wątek z wykorzystaniem prawa dostępu `DirectImpersonation`, możesz użyć innego mechanizmu. Dysponując uchwytem wątku źródłowego, możesz zastosować wywołanie systemowe `NtImpersonateThread` i przypisać token personifikacji innemu wątkowi. Skorzystanie z tego wywołania stanowi połączenie jawnej i niejawnej personifikacji. Jądro przechwyci kontekst personifikacji, tak jakby wątek źródłowy utworzył wywołanie za pośrednictwem nazwanego potoku. Możesz nawet określić strukturę opcji SQoS w ramach wywołania systemowego.

Możesz się zastanawiać, czy podszywanie się nie stwarza ogromnej luki w zabezpieczeniach. Jeśli skonfiguruję własny nazwany potok i przekonam proces uprzywilejowany do połączenia się z nim, a obiekt wywołujący nie ustawi opcji SQoS w celu ograniczenia dostępu, to czy nie uzyskam zwiększonych uprawnień? Kwestią zapobiegnięcia temu zajmiemy się ponownie w podrozdziale „Przypisywanie tokena" tego rozdziału.

# Przekształcanie typów tokenów

Możliwe jest przekształcenie jednego typu tokena w drugi z użyciem metody duplikacji. Gdy duplikujesz token, jądro tworzy nowy obiekt `Token` i sporządza rozbudowaną kopię wszystkich jego właściwości. W trakcie duplikowania tokena możesz zmienić jego typ.

Operacja duplikowania różni się od duplikacji uchwytu, którą omówiono w rozdziale 3., ponieważ duplikowanie uchwytu tokena spowodowałoby utworzenie jedynie nowego uchwytu wskazującego na ten sam obiekt `Token`. Aby zduplikować faktyczny obiekt `Token`, musisz mieć prawo dostępu `Duplicate` względem uchwytu.

W celu zduplikowania tokena możesz następnie użyć wywołania systemowego `NtDuplicateToken` lub polecenia `Copy-NtToken` narzędzia PowerShell. Aby na przykład na podstawie istniejącego tokena utworzyć token personifikacji na poziomie delegowania, skorzystaj ze skryptu z listingu 4.4.

*Listing 4.4. Duplikowanie tokena w celu utworzenia tokena personifikacji*

```
PS> $imp_token = Copy-NtToken -Token $token -ImpersonationLevel Delegation
PS> $imp_token.ImpersonationLevel
Delegation

PS> $imp_token.TokenType
Impersonation
```

Używając ponownie polecenia `Copy-NtToken`, token personifikacji można przekształcić z powrotem w token podstawowy (listing 4.5).

*Listing 4.5. Przekształcenie tokena personifikacji w token podstawowy*

```
PS> $pri_token = Copy-NtToken -Token $imp_token -Primary
PS> $pri_token.TokenType
Primary

PS> $pri_token.ImpersonationLevel
Delegation
```

Zwróć uwagę na coś interesującego w danych wyjściowych: nowy token podstawowy ma taki sam poziom personifikacji jak pierwotny token. Wynika to stąd, że monitor SRM bierze pod uwagę tylko właściwość `TokenType`. Jeśli token jest tokenem podstawowym, poziom personifikacji jest ignorowany.

Skoro można przekształcić token personifikacji z powrotem w token podstawowy, możesz zadać następujące pytanie: czy można by przekształcić token na poziomie `Identification` lub `Anonymous` z powrotem na token podstawowy, utworzyć nowy proces i obejść ustawienia opcji SQoS? Spróbujmy to sprawdzić za pomocą poleceń z listingu 4.6.

*Listing 4.6. Duplikowanie tokena na poziomie Identification z powrotem do postaci tokena podstawowego*

```
PS> $imp_token = Copy-NtToken -Token $token -ImpersonationLevel Identification
PS> $pri_token = Copy-NtToken -Token $imp_token -Primary
Exception: "(0xC00000A5) - A specified impersonation level is invalid."
```

Listing pokazuje, że nie można zduplikować tokena na poziomie `Identification` z powrotem do postaci tokena podstawowego. Drugi wiersz listingu powoduje wyjątek, ponieważ operacja ta naruszyłaby gwarancje monitora SRM dotyczące bezpieczeństwa (a dokładniej chodzi o to, że opcja SQoS pozwala obiektowi wywołującemu kontrolować to, jak używana jest jego tożsamość).

Ostatnia uwaga: jeśli otwierasz token przy użyciu polecenia `Get-NtToken`, podając parametr `Duplicate`, możesz wykonać operację duplikowania w jednym kroku.

# Pseudouchwyty tokenów

Aby uzyskać dostęp do tokena, musisz otworzyć uchwyt obiektu `Token`, a następnie pamiętać o zamknięciu uchwytu po jego użyciu. W systemie Windows 10 wprowadzono trzy *pseudouchwyty*, które pozwalają uzyskać informacje dotyczące tokenów bez otwierania pełnego uchwytu obiektu jądra. Oto te trzy uchwyty wraz z ich wartościami podanymi w nawiasach okrągłych:

**`Primary` (`-4`)**. Token podstawowy bieżącego procesu.

**`Impersonation` (`-5`)**. Token personifikacji bieżącego wątku. Operacja nie powiedzie się, jeśli wątek się nie podszywa.

**`Effective` (`-6`)**. Token personifikacji bieżącego wątku (jeśli się on podszywa). W przeciwnym razie jest to token podstawowy.

W przeciwieństwie do pseudouchwytów bieżącego procesu i wątku nie można duplikować tych uchwytów tokenów. Uchwytów można używać tylko w ograniczonej liczbie zastosowań, takich jak uzyskiwanie informacji lub przeprowadzanie kontroli dostępu. Jeżeli określisz parametr `Pseudo`, polecenie `Get-NtToken` może zwrócić te uchwyty (listing 4.7).

*Listing 4.7. Uzyskiwanie informacji o pseudouchwytach tokenów*

```
PS> Invoke-NtToken -Anonymous {Get-NtToken -Pseudo -Primary | Get-NtTokenSid}
Name                          Sid
----                          ---
GRAPHITE\user                 S-1-4-21-2318445812-3516008893-216915059-1002  (1)

PS> Invoke-NtToken -Anonymous {Get-NtToken -Pseudo -Impersonation | Get-NtTokenSid}
Name                          Sid
----                          ---
```

```
NT AUTHORITY\ANONYMOUS LOGON S-1-4-7  ❷

PS> Invoke-NtToken -Anonymous {Get-NtToken -Pseudo -Effective | Get-NtTokenSid}
Name                         Sid
----                         ---
NT AUTHORITY\ANONYMOUS LOGON S-1-4-7  ❸

PS> Invoke-NtToken -Anonymous {Get-NtToken -Pseudo -Effective} | Get-NtTokenSid
Name                         Sid
----                         ---
GRAPHITE\user                S-1-4-21-2318445812-3516008893-216915059-1002  ❹
```

W trakcie podszywania się pod użytkownika anonimowego utworzono tutaj zapytania dotyczące trzech typów pseudouchwytów tokenów. Pierwsze polecenie uzyskuje informacje o tokenie podstawowym i wyodrębnia identyfikator SID jego użytkownika ❶. Następne polecenie wysyła zapytanie dotyczące tokena personifikacji, które zwraca identyfikator SID użytkownika anonimowego ❷. Dalej zażądano efektywnego tokena, który również zwraca identyfikator SID użytkownika anonimowego, ponieważ podszyto się pod niego ❸. Na koniec ponownie utworzono zapytanie odnoszące się do efektywnego tokena, czekając tym razem na zakończenie wykonywania bloku skryptu, aby wyodrębnić identyfikator SID użytkownika. Operacja ta zwraca identyfikator SID użytkownika tokena podstawowego ❹, co pokazuje, że pseudouchwyt tokena jest kontekstowy.

# Grupy tokenów

Jeśli administratorzy musieliby zabezpieczać każdy zasób pod kątem każdego możliwego użytkownika, zabezpieczenia oparte na tożsamości stałyby się zbyt trudne do zarządzania. *Grupy* umożliwiają użytkownikom wspólne korzystanie z szerszej tożsamości zabezpieczeń. Większość operacji kontroli dostępu w systemie Windows przyznaje dostęp grupom, a nie poszczególnym użytkownikom.

Z perspektywy monitora SRM grupa to po prostu kolejny identyfikator SID, który potencjalnie może zdefiniować dostęp do zasobu. Grupy można wyświetlić w konsoli narzędzia PowerShell za pomocą polecenia `Get-NtTokenGroup` (listing 4.8).

*Listing 4.8. Uzyskiwanie grup bieżącego tokena*

```
PS> Get-NtTokenGroup $token
Name                                Attributes
----                                ----------
GRAPHITE\None                       Mandatory, EnabledByDefault, Enabled
Everyone                            Mandatory, EnabledByDefault, Enabled
BUILTIN\Users                       Mandatory, EnabledByDefault, Enabled
BUILTIN\Performance Log Users       Mandatory, EnabledByDefault, Enabled
NT AUTHORITY\INTERACTIVE            Mandatory, EnabledByDefault, Enabled
--cięcie--
```

Możliwe jest również użycie polecenia `Get-NtTokenGroup` z podanym parametrem `Attributes`, służącego do filtrowania pod kątem konkretnych flag atrybutu. W tabeli 4.2 zaprezentowano możliwe flagi atrybutu, które można przekazać poleceniu.

*Tabela 4.2. Atrybuty grup w formacie zestawu SDK i narzędzia PowerShell*

| Nazwa atrybutu zestawu SDK | Nazwa atrybutu narzędzia PowerShell |
|---|---|
| SE_GROUP_ENABLED | Enabled |
| SE_GROUP_ENABLED_BY_DEFAULT | EnabledByDefault |
| SE_GROUP_MANDATORY | Mandatory |
| SE_GROUP_LOGON_ID | LogonId |
| SE_GROUP_OWNER | Owner |
| SE_GROUP_USE_FOR_DENY_ONLY | UseForDenyOnly |
| SE_GROUP_INTEGRITY | Integrity |
| SE_GROUP_INTEGRITY_ENABLED | IntegrityEnabled |
| SE_GROUP_RESOURCE | Resource |

W dalszej części rozdziału opisano znaczenie każdej z tych flag.

## Flagi Enabled, EnabledByDefault i Mandatory

Najważniejsza flaga to `Enabled`. Po ustawieniu jej monitor SRM bierze pod uwagę grupę podczas procesu kontroli dostępu. W przeciwnym razie zignoruje grupę. Każda grupa z ustawioną flagą `EnabledByDefault` jest automatycznie włączona.

Jeśli dysponujesz prawem dostępu `AdjustGroups` względem uchwytu tokena, możliwe jest wyłączenie grupy (wykluczenie jej z procesu kontroli dostępu) za pomocą wywołania systemowego `NtAdjustGroupsToken`. Polecenie `Set-NtTokenGroup` narzędzia PowerShell udostępnia to wywołanie. Nie możesz jednak wyłączyć grup z ustawioną flagą `Mandatory`. Flaga jest ustawiona dla wszystkich grup w tokenie zwykłego konta użytkownika, ale niektóre tokeny systemowe mają grupy, które nie są obowiązkowe. Jeżeli grupa jest wyłączona w momencie przekazywania tokena personifikacji za pośrednictwem wywołania RPC, a w opcji SQoS ustawiono flagę trybu efektywnego tokena, token personifikacji usunie grupę.

## Flaga LogonId

Flaga `LogonId` identyfikuje dowolny identyfikator SID nadany wszystkim tokenom na tym samym pulpicie. Jeśli na przykład uruchomisz proces jako inny użytkownik przy użyciu narzędzia `runas`, token nowego procesu będzie miał ten sam identyfikator SID logowania co obiekt wywołujący nawet mimo tego, że to inna tożsamość. Taki sposób działania pozwala monitorowi SRM na przyznanie dostępu do zasobów powiązanych z sesją, takich jak katalog obiektów sesji. Identyfikator SID zawsze ma format `S-1-4-4-`*`X`*`-`*`Y`*, gdzie *`X`* i *`Y`* to dwie 32-bitowe wartości identyfikatora LUID, które zostały przydzielone w momencie utworzenia sesji uwierzytelnienia. Do kwestii identyfikatora SID logowania i obszarów jego zastosowania powrócimy w następnym rozdziale.

## Flaga Owner

Wszystkie zasoby w systemie, które mogą zostać zabezpieczone, należą do identyfikatora SID grupy lub użytkownika. Tokeny uwzględniają właściwość `Owner` zawierającą identyfikator SID używany podczas tworzenia zasobu jako domyślny właściciel. Monitor SRM zezwala tylko na określenie we właściwości `Owner` konkretnego zestawu identyfikatorów SID użytkowników: albo identyfikatora SID użytkownika, albo identyfikatora SID dowolnej grupy oznaczonego flagą `Owner`.

Za pomocą polecenia `Get-NtTokenSid` lub `Set-NtTokenSid` możesz odpowiednio uzyskać lub ustawić bieżącą wartość właściwości `Owner` tokena. Na przykład w listingu 4.9 uzyskano z bieżącego tokena identyfikator SID właściciela, a następnie podjęto próbę ustawienia właściciela.

*Listing 4.9. Uzyskiwanie i ustawianie identyfikatora SID właściciela tokena*

```
PS> Get-NtTokenSid $token -Owner
Name          Sid
----          ---
GRAPHITE\user S-1-4-21-818064984-378290696-2985406761-1002

PS> Set-NtTokenSid -Owner -Sid "S-1-2-3-4"
Exception setting "Owner": "(0xC000005A) - Indicates a particular
Security ID may not be assigned as the owner of an object."
```

W tym przypadku próba ustawienia dla właściwości `Owner` identyfikatora SID `S-1-2-3-4` kończy się niepowodzeniem i zgłoszeniem wyjątku, ponieważ nie jest to identyfikator SID bieżącego użytkownika ani żadnej grupy znajdującej się na liście.

## Flaga UseForDenyOnly

Kontrola dostępu przeprowadzana przez monitor SRM przyznaje dostęp do identyfikatora SID lub odmawia go. Gdy jednak wyłączono identyfikator SID, nie będzie on już uwzględniany w procesie kontroli udzielającym dostępu lub odmawiającym go, co może skutkować jego niepoprawnością.

Rozważmy prosty przykład. Wyobraź sobie, że istnieją dwie grupy: *Employee* (pracownik) i *Remote Access* (zdalny dostęp). Użytkownik tworzy dokument i oczekuje, że będą mogli go odczytać wszyscy pracownicy, z wyjątkiem tych uzyskujących zdalnie dostęp do systemu, ponieważ zawartość dokumentu jest poufna, a użytkownik nie chce, aby informacje zostały ujawnione. Dokument jest tak skonfigurowany, aby przyznawać dostęp wszystkim członkom grupy *Employee*, lecz odmówić dostępu użytkownikom z grupy *Remote Access*.

Wyobraź sobie teraz, że użytkownik należący do obu tych grup mógłby wyłączyć grupę podczas uzyskiwania dostępu do zasobu. Mógłby on po prostu wyłączyć grupę *Remote Access*, żeby dostęp do dokumentu był przyznawany danej osobie na podstawie jej członkostwa w grupie *Employee*, co oznaczałoby obejście z łatwością restrykcji dotyczących dostępu.

Z tego powodu użytkownik rzadko będzie miał możliwość wyłączenia grup. W niektórych jednak przypadkach, takich jak izolowanie w „piaskownicy", będzie wskazana

możliwość wyłączenia grupy, aby nie mogła być użyta do uzyskania dostępu do zasobu. Flaga `UseForDenyOnly` rozwiązuje ten problem. Gdy identyfikator SID oznaczono tą flagą, nie będzie brany pod uwagę podczas sprawdzania możliwości uzyskania dostępu, ale nadal będzie uwzględniany w trakcie kontroli dostępu dotyczącej jego odmowy. Użytkownik może oznaczyć własne grupy flagą `UseForDenyOnly` przez odfiltrowanie swojego tokena i użycie go do utworzenia nowego procesu. Filtrowanie tokenów zostanie omówione w kontekście ograniczonych tokenów w podrozdziale „Tokeny »piaskownicy«" niniejszego rozdziału.

## Flagi Integrity i IntegrityEnabled

Flagi atrybutu `Integrity` i `IntegrityEnabled` są włączone i wskazują, że identyfikator SID reprezentuje poziom integralności tokena. Identyfikatory SID grup oznaczone flagą atrybutu `Integrity` przechowują ten poziom integralności w swoim końcowym identyfikatorze RID jako liczbę 32-bitową. Identyfikator RID może mieć dowolną wartość. W zestawie SDK zdefiniowano wstępnie jednak siedem poziomów zaprezentowanych w tabeli 4.3. Tylko pierwsze sześć z nich jest powszechnie używanych i dostępnych z poziomu procesu użytkownika. Aby wskazać identyfikator SID integralności, monitor SRM używa ustawienia `MandatoryLabel` jednostki autoryzacji zabezpieczeń (o wartości `16`).

*Tabela 4.3. Wartości predefiniowanych poziomów integralności*

| Poziom integralności | Nazwa z zestawu SDK | Nazwa z narzędzia PowerShell |
|---|---|---|
| 0 | SECURITY_MANDATORY_UNTRUSTED_RID | Untrusted |
| 4096 | SECURITY_MANDATORY_LOW_RID | Low |
| 8192 | SECURITY_MANDATORY_MEDIUM_RID | Medium |
| 8448 | SECURITY_MANDATORY_MEDIUM_PLUS_RID | MediumPlus |
| 12288 | SECURITY_MANDATORY_HIGH_RID | High |
| 16384 | SECURITY_MANDATORY_SYSTEM_RID | System |
| 20480 | SECURITY_MANDATORY_PROTECTED_PROCESS_RID | ProtectedProcess |

W przypadku użytkownika domyślny poziom to `Medium`. Administratorzy mają zwykle przypisany poziom `High`, a usługi poziom `System`. Używając polecenia `Get-NtTokenSid`, można sprawdzić identyfikator SID integralności tokena (listing 4.10).

*Listing 4.10. Uzyskiwanie identyfikatora SID integralności tokena*

```
PS> Get-NtTokenSid $token -Integrity
Name                                Sid
----                                ---

Mandatory Label\Medium Mandatory Level S-1-16-8192
```

Można również ustawić nowy poziom integralności tokena, pod warunkiem że jego wartość jest mniejsza lub równa bieżącej wartości. Możliwe jest także zwiększenie poziomu, ale wymaga to specjalnych uprawnień oraz włączenia uprawnienia `SeTcbPrivilege`.

Choć można ustawić cały identyfikator SID, zwykle wygodniej jest ustawić tylko wartość. Na przykład skrypt z listingu 4.11 ustawia poziom integralności tokena na `Low`.

*Listing 4.11. Ustawienie poziomu integralności tokena na Low*

```
PS> Set-NtTokenIntegrityLevel Low -Token $token
PS> Get-NtTokenSid $token -Integrity
Name                                Sid
----                                ---
Mandatory Label\Low Mandatory Level S-1-16-4096
```

Jeśli uruchomisz ten skrypt, może się okazać, że zaczniesz uzyskiwać błędy w konsoli narzędzia PowerShell związane z zablokowanym dostępem do plików. W trakcie omawiania w rozdziale 7. mechanizmu obowiązkowej kontroli integralności MIC (*Mandatory Integrity Control*) zostanie wyjaśnione, dlaczego blokowany jest dostęp do plików.

## Flaga Resource

Ostatnia flaga atrybutu zasługuje jedynie na krótką wzmiankę. Flaga `Resource` wskazuje, że identyfikator SID grupy jest *lokalnym identyfikatorem SID w domenie*. Powrócimy do tego typu identyfikatora SID w rozdziale 10.

## Grupy urządzeń

Z tokenem może być też związana oddzielna lista *grup urządzeń*. Identyfikatory SID tych grup są dodawane, gdy użytkownik uwierzytelnia się na serwerze za pośrednictwem sieci w środowisku przedsiębiorstwa (listing 4.12).

*Listing 4.12. Wyświetlanie grup urządzeń za pomocą polecenia Get-NtTokenGroup*

```
PS> Get-NtTokenGroup -Device -Token $token
Name                              Attributes
----                              ----------
BUILTIN\Users                     Mandatory, EnabledByDefault, Enabled
AD\CLIENT1$                       Mandatory, EnabledByDefault, Enabled
AD\Domain Computers               Mandatory, EnabledByDefault, Enabled
NT AUTHORITY\Claims Value         Mandatory, EnabledByDefault, Enabled
--cięcie--
```

Korzystając z polecenia `Get-NtTokenGroup` i przekazując parametr `Device`, możesz uzyskać grupy powiązane z tokenem.

# Uprawnienia

Grupy pozwalają administratorom systemu kontrolować dostęp użytkownika do określonych zasobów. Z kolei *uprawnienia* są przyznawane użytkownikowi, aby umożliwić mu uproszczenie niektórych kontroli zabezpieczeń w przypadku wszelkich typów zasobów (np. przez ominięcie kontroli dostępu). Uprawnienie może także dotyczyć określonych działań uprzywilejowanych, takich jak zmiana zegara systemowego. Używając polecenia `Get-NtTokenPrivilege`, w konsoli możesz wyświetlić uprawnienia tokena (listing 4.13).

*Listing 4.13. Wyszczególnienie uprawnień tokena*

```
PS> Get-NtTokenPrivilege $token
Name                           Luid                Enabled
----                           ----                -------
SeShutdownPrivilege            00000000-00000013   False
SeChangeNotifyPrivilege        00000000-00000017   True
SeUndockPrivilege              00000000-00000019   False
SeIncreaseWorkingSetPrivilege  00000000-00000021   False
SeTimeZonePrivilege            00000000-00000022   False
```

Dane wynikowe podzielono na trzy kolumny. Pierwsza kolumna to powszechna nazwa uprawnienia. Podobnie jak w przypadku identyfikatorów SID, monitor SRM nie korzysta bezpośrednio z tej nazwy. Zamiast tego używa wartości identyfikatora LUID uprawnienia, którą można zobaczyć w drugiej kolumnie. Ostatnia kolumna wskazuje, czy uprawnienie jest obecnie włączone. Uprawnienia mogą być w stanie włączonym lub wyłączonym.

Każde sprawdzenie uprawnienia powinno zapewnić, że jest ono włączone, a nie tylko obecne. W pewnych okolicznościach, takich jak izolowanie w „piaskownicy", token może zawierać wyszczególnione uprawnienie, ale ograniczenia „piaskownicy" mogą uniemożliwiać oznaczenie go jako włączonego. Flaga `Enabled` to w rzeczywistości zbiór flag atrybutów podobnie jak atrybuty identyfikatorów SID grup. Atrybuty te można ujrzeć, formatując dane wynikowe polecenia `Get-NtTokenPrivilege` jako listę (listing 4.14).

*Listing 4.14. Wyświetlenie wszystkich właściwości uprawnienia SeChangeNotifyPrivilege*

```
PS> Get-NtTokenPrivilege $token -Privileges
SeChangeNotifyPrivilege | Format-List
Name        : SeChangeNotifyPrivilege
Luid        : 00000000-00000017
Attributes  : EnabledByDefault, Enabled
Enabled     : True
DisplayName : Bypass traverse checking
```

W danych wynikowych można teraz dostrzec atrybuty, które obejmują zarówno atrybut `Enabled`, jak i atrybut `EnabledByDefault`. Atrybut `EnabledByDefault` określa, czy domyślny stan uprawnienia to stan włączenia. Na tym etapie można również zobaczyć dodatkową właściwość `DisplayName`, która dostarcza użytkownikowi kolejnych informacji.

Aby zmodyfikować stan uprawnień tokena, wymagasz prawa dostępu `AdjustPrivileges` względem uchwytu tokena. W dalszej kolejności możesz użyć wywołania systemowego

NtAdjustPrivilegesToken, aby dostosować atrybuty oraz włączyć lub wyłączyć uprawnienie. Polecenia Enable-NtTokenPrivilege i Disable-NtTokenPrivilege narzędzia PowerShell udostępniają to wywołanie (listing 4.15).

*Listing 4.15. Włączanie i wyłączanie uprawnienia SeTimeZonePrivilege*

```
PS> Enable-NtTokenPrivilege SeTimeZonePrivilege -Token $token -PassThru
Name                           Luid                Enabled
----                           ----                -------
SeTimeZonePrivilege            00000000-00000022   True

PS> Disable-NtTokenPrivilege SeTimeZonePrivilege -Token $token -PassThru
Name                           Luid                Enabled
----                           ----                -------
SeTimeZonePrivilege            00000000-00000022   False
```

Korzystając z interfejsu API NtAdjustPrivilegesToken, można też całkowicie usunąć uprawnienie przez ustawienie atrybutu Remove, co można zrealizować za pomocą polecenia Remove-NtTokenPrivilege narzędzia PowerShell. Usunięcie uprawnienia zapewnia, że token nigdy nie będzie mógł go ponownie użyć. Jeśli tylko wyłączysz uprawnienie, mogłoby ono zostać nieświadomie ponownie włączone. Listing 4.16 pokazuje, jak usunąć uprawnienie.

*Listing 4.16. Usuwanie uprawnienia z tokena*

```
PS> Get-NtTokenPrivilege $token -Privileges SeTimeZonePrivilege
Name                           Luid                Enabled
----                           ----                -------
SeTimeZonePrivilege 00000000-00000022 False

PS> Remove-NtTokenPrivilege SeTimeZonePrivilege -Token $token
PS> Get-NtTokenPrivilege $token -Privileges SeTimeZonePrivilege
WARNING: Couldn't get privilege SeTimeZonePrivilege
```

Aby sprawdzić uprawnienia, aplikacja użytkownika może zastosować wywołanie systemowe NtPrivilegeCheck, natomiast kod jądra może wywołać interfejs API SePrivilege Check. Możesz się zastanawiać, czy możesz po prostu ręcznie sprawdzić, czy uprawnienie jest włączone, zamiast korzystać ze specjalnego wywołania systemowego. W tym przypadku tak jest. Zawsze warto jednak korzystać z systemowych rozwiązań, gdy tylko jest to możliwe, aby uniknąć konsekwencji błędu w implementacji lub nieprzewidzianego, skrajnego przypadku. Polecenie Test-NtTokenPrivilege narzędzia PowerShell opakowuje wywołanie systemowe (listing 4.17).

*Listing 4.17. Wykonywanie operacji sprawdzania uprawnień*

```
PS> Enable-NtTokenPrivilege SeChangeNotifyPrivilege
PS> Disable-NtTokenPrivilege SeTimeZonePrivilege
PS> Test-NtTokenPrivilege SeChangeNotifyPrivilege
True
```

```
PS> Test-NtTokenPrivilege SeTimeZonePrivilege, SeChangeNotifyPrivilege -All
False

PS> Test-NtTokenPrivilege SeTimeZonePrivilege, SeChangeNotifyPrivilege
-All -PassResult
EnabledPrivileges           AllPrivilegesHeld
-----------------           -----------------
{SeChangeNotifyPrivilege}   False
```

Listing demonstruje przykład sprawdzania uprawnień przy użyciu polecenia `Test-NtTokenPrivilege`. Najpierw włączono uprawnienie `SeChangeNotifyPrivilege` i wyłączono uprawnienie `SeTimeZonePrivilege`. Są to typowe uprawnienia przyznawane wszystkim użytkownikom, ale może być konieczna modyfikacja przykładu, jeśli Twój token jest ich pozbawiony.

Dalej dokonano sprawdzenia dotyczącego wyłącznie uprawnienia `SeChangeNotifyPrivilege`. Jest ono włączone, dlatego test zwraca wartość `True`. Sprawdzono następnie zarówno uprawnienie `SeTimeZonePrivilege`, jak i uprawnienie `SeChangeNotifyPrivilege`. Widać, że nie są dostępne wszystkie uprawnienia, stąd też polecenie `Test-NtTokenPrivilege` zwraca wartość `False`. Na koniec wykonano to samo polecenie, ale z opcją `-PassResult`, aby zwrócić pełny wynik sprawdzenia. W kolumnie `EnabledPrivileges` widać, że włączone jest tylko uprawnienie `SeChangeNotifyPrivilege`.

Oto niektóre z dostępnych uprawnień systemowych:

**`SeChangeNotifyPrivilege`**. Nazwa tego uprawnienia jest myląca. Umożliwia ono użytkownikowi otrzymywanie powiadomień o zmianach w systemie plików lub rejestrze, ale pozwala także pominąć sprawdzanie uprawnień zezwalających na operację przemieszczania się między katalogami (zostanie ona omówiona w rozdziale 8.).

**`SeAssignPrimaryTokenPrivilege` i `SeImpersonatePrivilege`**. Uprawnienia pozwalają użytkownikowi na pominięcie odpowiednio kontroli przypisywania tokena podstawowego i kontroli personifikacji (podszywania się). W przeciwieństwie do większości uprawnień na tej liście, te dwa uprawnienia muszą być włączone w tokenie podstawowym bieżącego procesu, a nie w tokenie personifikacji.

**`SeBackupPrivilege` i `SeRestorePrivilege`**. Uprawnienia umożliwiają użytkownikowi pominięcie sprawdzania dostępu podczas otwierania określonych zasobów, takich jak pliki lub klucze rejestru. Dzięki temu użytkownik może tworzyć kopie zapasowe i przywracać zasoby bez konieczności jawnego przyznawania dostępu do nich. Uprawnienia te są także używane do innych celów: na przykład uprawnienie przywracania pozwala użytkownikowi załadować dowolne gałęzi rejestru.

**`SeSecurityPrivilege` i `SeAuditPrivilege`**. Pierwsze z tych uprawnień daje użytkownikowi możliwość uzyskania prawa dostępu `AccessSystemSecurity` do zasobu, co pozwala na modyfikację jego konfiguracji audytu. Uprawnienie `SeAuditPrivilege` umożliwia użytkownikowi generowanie z poziomu aplikacji użytkownika dowolnych komunikatów związanych z audytem obiektów. Kwestia audytu zostanie omówiona w rozdziałach 5., 6. i 9.

**SeCreateTokenPrivilege**. Uprawnienie powinno być przyznawane tylko bardzo wyselekcjonowanej grupie użytkowników, ponieważ umożliwia tworzenie dowolnych tokenów za pomocą wywołania systemowego `NtCreateToken`.

**SeDebugPrivilege**. Nazwa tego uprawnienia sugeruje, że jest ono niezbędne do debugowania procesów. W rzeczywistości nie jest to jednak prawda, gdyż można debugować proces bez tego uprawnienia. Pozwala ono użytkownikowi pominąć dowolne sprawdzenie dostępu podczas otwierania obiektu procesu lub wątku.

**SeTcbPrivilege**. Łańcuch `Tcb` w nazwie tego uprawnienia jest skrótem od terminu *Trusted Computing Base* (zaufana baza obliczeniowa). Odnosi się on do uprzywilejowanej, centralnej części systemu operacyjnego Windows obejmującej jądro. Uprawnienie to uwzględnia uprzywilejowane operacje, które nie są objęte zakresem bardziej specyficznego uprawnienia. Uprawnienie umożliwia na przykład użytkownikom pominięcie kontroli dotyczącej zwiększania poziomu integralności tokena (aż do granicznego poziomu `System`). Pozwala też określić dla procesu awaryjną procedurę obsługi wyjątku. Te dwie operacje nie mają ze sobą wiele wspólnego.

**SeLoadDriverPrivilege**. Możliwe jest załadowanie nowego sterownika jądra za pomocą wywołania systemowego `NtLoadDriver`, choć częściej używa się do tego monitora SCM. Uprawnienie to jest wymagane do pomyślnego zastosowania tego wywołania. Zauważ, że posiadanie tego uprawnienia nie pozwala na ominięcie operacji sprawdzania sterowników jądra, takich jak podpisywanie kodu.

**SeTakeOwnershipPrivilege i SeRelabelPrivilege**. Uprawnienia powodują ten sam natychmiastowy efekt, czyli pozwalają użytkownikowi uzyskać prawo dostępu `WriteOwner` do zasobu, jeśli nawet standardowa kontrola dostępu nie umożliwiłaby tego. Uprawnienie `SeTakeOwnershipPrivilege` daje użytkownikowi możliwość przejęcia własności zasobu, ponieważ do zrealizowania tego celu niezbędne jest posiadanie prawa dostępu `WriteOwner`. Zastosowanie uprawnienia `SeRelabelPrivilege` powoduje pominięcie sprawdzeń etykiety zasobu związanej z obowiązkową kontrolą. Standardowo możesz ustawić jedynie etykietę z poziomem równym poziomowi integralności obiektu wywołującego lub niższym. Określenie etykiety obowiązkowej kontroli wymaga również prawa dostępu `WriteOwner` względem uchwytu, o czym będzie mowa w rozdziale 6.

W dalszych rozdziałach zostaną przedstawione konkretne przykłady użycia tych uprawnień w trakcie omawiania deskryptorów zabezpieczeń i sprawdzania dostępu. Na razie zajmijmy się metodami ograniczania dostępu za pośrednictwem izolowania w „piaskownicy”.

# Tokeny „piaskownicy”

W naszym połączonym świecie trzeba przetwarzać mnóstwo danych, którym się nie ufa. Atakujący mogą preparować dane w złych zamiarach, takich jak wykorzystanie luki w zabezpieczeniach przeglądarki internetowej lub czytnika dokumentów. Aby przeciwdziałać temu zagrożeniu, system Windows oferuje metodę ograniczenia zasobów, do których

użytkownik może uzyskać dostęp, przez umieszczanie w „piaskownicy” wszelkich jego procesów obsługujących niezaufane dane. Jeśli zabezpieczenia procesu zostaną naruszone, atakujący będzie miał jedynie ograniczony wgląd w system i nie będzie w stanie uzyskać dostępu do poufnych informacji użytkownika. System Windows implementuje „piaskownice” za pomocą trzech specjalnych typów tokenów: tokenów ograniczonych, tokenów z ograniczeniem zapisu oraz tokenów *lowbox*.

## Tokeny ograniczone

*Token ograniczony* (ang. *restricted token*) to najstarszy w systemie Windows typ tokena „piaskownicy”. Wprowadzono go jako opcję w systemie Windows 2000. Token nie był jednak szeroko stosowany jako „piaskownica” aż do czasu pojawienia się przeglądarki internetowej Google Chrome. W innych przeglądarkach, takich jak Firefox, powielono w tamtym czasie implementację „piaskownicy” przeglądarki Chrome. Podobnie stało się w przypadku programów do czytania dokumentów, jak np. Adobe Reader.

Token ograniczony możesz utworzyć za pomocą wywołania systemowego `NtFilterToken` lub interfejsu API `CreateRestrictedToken` podsystemu Win32. Umożliwiają one określenie listy ograniczonych identyfikatorów SID w celu ograniczenia zasobów, do jakich token będzie mieć dostęp. Identyfikatory SID nie muszą być już obecne w tokenie. Na przykład najbardziej restrykcyjna „piaskownica” przeglądarki Chrome określa identyfikator SID NULL (`S-1-0-0`) jako jedyny ograniczony identyfikator. Nigdy nie jest on przypisywany do tokena jako zwykła grupa.

Każda kontrola dostępu musi uwzględniać zarówno standardową listę grup, jak i listę ograniczonych identyfikatorów SID. W przeciwnym razie użytkownikowi zostanie odmówiony dostęp, co zostanie szczegółowo omówione w rozdziale 7. Wywołanie systemowe `NtFilterToken` także może oznaczać zwykłe grupy za pomocą flagi atrybutu `UseForDenyOnly` i usuwać uprawnienia. Możliwość filtrowania tokena można połączyć z ograniczonymi identyfikatorami SID lub użyć go samodzielnie w celu uzyskania tokena o mniejszych uprawnieniach bez bardziej rozbudowanego izolowania w „piaskownicy”.

Łatwo jest utworzyć token ograniczony, który nie ma dostępu do żadnych zasobów. Takie ograniczenie zapewnia odpowiednią „piaskownicę”, ale sprawia także, że niemożliwe jest wykorzystanie tokena jako tokena podstawowego procesu, ponieważ nie będzie możliwe uruchomienie go. To poważnie ogranicza poziom skuteczności „piaskownicy” korzystającej z tokenów ograniczonych. Listing 4.18 demonstruje, jak utworzyć token ograniczony i wyodrębnić wyniki.

*Listing 4.18. Tworzenie tokena ograniczonego oraz wyświetlanie grup i uprawnień*

```
PS> $token = Get-NtToken -Filtered -RestrictedSids RC -SidsToDisable WD
-Flags DisableMaxPrivileges
PS> Get-NtTokenGroup $token -Attributes UseForDenyOnly
Name                           Attributes
----                           ----------
Everyone                       UseForDenyOnly

PS> Get-NtTokenGroup $token -Restricted
Name                           Attributes
----                           ----------
```

```
NT AUTHORITY\RESTRICTED        Mandatory, EnabledByDefault, Enabled

PS> Get-NtTokenPrivilege $token
Name                           Luid                 Enabled
----                           ----                 -------
SeChangeNotifyPrivilege        00000000-00000017    True

PS> $token.Restricted
True
```

Zaczęto od utworzenia tokena ograniczonego za pomocą polecenia `Get-NtToken`. Określono jeden ograniczony identyfikator SID `RC`, który jest mapowany na specjalny identyfikator SID `NT AUTHORITY\RESTRICTED` konfigurowany często dla zasobów systemowych w celu umożliwienia dostępu z odczytem. Ponadto zdefiniowano, że grupa *Everyone* (`WD`) ma zostać przekształcona w atrybut `UseForDenyOnly`. Na koniec określono flagę powodującą wyłączenie wariantu maksymalnej liczby uprawnień.

Korzystając z atrybutu `UseForDenyOnly`, wyświetlono następnie właściwości tokena, zaczynając od wszystkich zwykłych grup. Dane wyjściowe potwierdzają, że tylko grupa *Everyone* ma ustawioną flagę. W dalszej kolejności wyświetlono listę ograniczonych identyfikatorów SID, która zawiera identyfikator `NT AUTHORITY\RESTRICTED`.

Dalej wyświetlono uprawnienia. Zauważ, że mimo tego, że zażądano wyłączenia maksymalnej liczby uprawnień, uprawnienie `SeChangeNotifyPrivilege` nadal pozostaje. Nie jest ono usuwane, ponieważ bez niego uzyskanie dostępu do zasobów może okazać się bardzo trudne. Jeśli naprawdę chcesz pozbyć się tego uprawnienia, możesz określić to wprost za pomocą wywołania systemowego `NtFilterToken` lub usunąć je po utworzeniu tokena.

Na koniec sprawdzono właściwość tokena, która wskazuje, czy jest on tokenem ograniczonym.

### TRYB CHRONIONY PRZEGLĄDARKI INTERNET EXPLORER

Program Internet Explorer 7 wprowadzony w systemie Windows Vista był pierwszą przeglądarką internetową w systemie Windows, która zawierała „piaskownicę”. Przeglądarka ta korzystała z możliwości obniżenia poziomu integralności tokena procesu w celu ograniczenia liczby zasobów, w przypadku których przeglądarka mogła dokonywać zapisu. Ostatecznie w systemie Windows 8 zastąpiono tę prostą „piaskownicę”, którą nazywano *trybem chronionym*, nowym typem tokena *lowbox*, który zostanie omówiony w punkcie „AppContainer i tokeny lowbox” tego rozdziału. Token *lowbox* zapewnił większą izolację (zwaną *rozszerzonym trybem chronionym*). Godne uwagi jest to, że firma Microsoft nie skorzystała z tokenów ograniczonych nawet pomimo tego, że były one dostępne od czasów systemu Windows 2000.

## Tokeny z ograniczeniem zapisu

*Token z ograniczeniem zapisu* (ang. *write-restricted token*) uniemożliwia dokonanie zapisu w zasobie, ale zezwala na dostęp z uprawnieniami odczytu i wykonywania. Token z ograniczeniem zapisu możesz utworzyć, przekazując flagę `WRITE_RESTRICTED` wywołaniu systemowemu `NtFilterToken`.

W systemie Windows XP SP2 wprowadzono ten typ tokena, aby zabezpieczyć usługi systemowe. Jest on znacznie łatwiejszy do zastosowania w roli „piaskownicy" niż token ograniczony, ponieważ nie musisz się martwić tym, że token nie będzie mógł odczytywać kluczowych zasobów, takich jak biblioteki DLL. Token z ograniczeniem zapisu tworzy jednak mniej przydatną „piaskownicę". Jeśli na przykład możesz odczytywać pliki użytkownika, możesz być w stanie wykraść jego prywatne informacje, takie jak hasła przechowywane przez przeglądarkę internetową, bez potrzeby opuszczania „piaskownicy".

Aby zapewnić kompletność informacji, utwórzmy token z ograniczeniem zapisu i sprawdźmy jego właściwości (listing 4.19).

*Listing 4.19. Tworzenie tokena z ograniczeniem zapisu*

```
PS> $token = Get-NtToken -Filtered -RestrictedSids WR -Flags WriteRestricted
PS> Get-NtTokenGroup $token -Restricted
Name                           Attributes
----                           ----------
NT AUTHORITY\WRITE RESTRICTED  Mandatory, EnabledByDefault, Enabled

PS> $token.Restricted
True

PS> $token.WriteRestricted
True
```

Zaczęto od utworzenia tokena za pomocą polecenia `Get-NtToken`. Określono jeden ograniczony identyfikator SID `WR`, który jest mapowany na specjalny identyfikator SID `NT AUTHORITY\WRITE RESTRICTED` będący odpowiednikiem identyfikatora `NT AUTHORITY\RESTRICTED`, ale udzielono dostępu z uprawnieniem zapisu do konkretnych zasobów systemowych. Dodatkowo użyto flagi `WriteRestricted`, aby utworzyć token z ograniczeniem zapisu zamiast zwykłego tokena ograniczonego.

W dalszej kolejności wyświetlono właściwości tokena. Na liście ograniczonych identyfikatorów SID widoczny jest identyfikator `NT AUTHORITY\WRITE RESTRICTED`. Wyświetlenie właściwości `Restricted` pozwala stwierdzić, że token jest uznawany za ograniczony. Widać również, że jest oznaczony jako uwzględniający flagę `WriteRestricted`.

## AppContainer i tokeny lowbox

W systemie Windows 8 wprowadzono „piaskownicę" AppContainer, aby chronić nowy model aplikacji systemu Windows. „Piaskownica" ta implementuje swoje zabezpieczenia za pomocą *tokena lowbox* (ang. *lowbox token*). Token ten możesz utworzyć na bazie istniejącego tokena za pomocą wywołania systemowego `NtCreateLowBoxToken`. Nie ma

bezpośredniego odpowiednika tego wywołania w postaci interfejsu API podsystemu Win32, ale można tworzyć proces „piaskownicy" AppContainer za pomocą interfejsu API `CreateProcess`. Nie będzie tutaj dokładniej omawiane, jak utworzyć proces za pomocą tego interfejsu API. Zamiast tego skupimy się wyłącznie na tokenie *lowbox*.

Podczas tworzenia tokena *lowbox* musisz określić identyfikator SID pakietu oraz listę identyfikatorów SID możliwości dostępu (ang. *capability SID*). Oba typy tych identyfikatorów są zapewniane przez *jednostkę autoryzacji pakietów aplikacji* (z przypisaną wartością 15). Identyfikatory SID pakietów możesz odróżnić od identyfikatorów SID możliwości dostępu przez sprawdzenie ich pierwszych identyfikatorów RID, które powinny wynosić odpowiednio 2 i 3. Identyfikator SID pakietu działa podobnie jak identyfikator SID użytkownika w zwykłym tokenie, natomiast identyfikatory SID możliwości dostępu funkcjonują jak ograniczone identyfikatory SID. Szczegóły dotyczące tego, jak te identyfikatory SID wpływają na proces sprawdzania dostępu, zostaną przedstawione w rozdziale 7.

Identyfikatory SID możliwości dostępu modyfikują razem proces sprawdzania dostępu, ale mogą też mieć jakieś znaczenie niezależnie od siebie. Istnieją na przykład możliwości pozwalające na uzyskanie dostępu sieciowego, które są obsługiwane specjalnie przez komponent zapory systemu Windows nawet pomimo tego, że nie są bezpośrednio powiązane z kontrolą dostępu. Istnieją następujące dwa typy identyfikatorów SID możliwości dostępu:

**Starsze**. Niewielki zestaw wstępnie zdefiniowanych identyfikatorów SID wprowadzonych w systemie Windows 8.

**Nazwane**. Identyfikatory RID generowane na podstawie nazwy tekstowej.

W dodatku B zamieszczono bardziej rozbudowaną listę nazwanych identyfikatorów SID możliwości dostępu. W tabeli 4.4 zaprezentowano starsze identyfikatory możliwości dostępu.

*Tabela 4.4. Starsze identyfikatory SID możliwości dostępu*

| Nazwa możliwości dostępu | Identyfikator SID |
|---|---|
| *Your internet connection* (Twoje połączenie internetowe) | S-1-15-3-1 |
| *Your internet connection, including incoming connections from the internet* (Twoje połączenie internetowe, w tym połączenia przychodzące z internetu) | S-1-15-3-2 |
| *Your home or work networks* (Twoje sieci domowe lub firmowe) | S-1-15-3-3 |
| *Your pictures library* (Twoja biblioteka zdjęć) | S-1-15-3-4 |
| *Your videos library* (Twoja biblioteka wideo) | S-1-15-3-5 |
| *Your music library* (Twoja biblioteka muzyki) | S-1-15-3-6 |
| *Your documents library* (Twoja biblioteka dokumentów) | S-1-15-3-7 |
| *Your Windows credentials* (Twoje dane uwierzytelniające w systemie Windows) | S-1-15-3-8 |

*Tabela 4.4. Starsze identyfikatory SID możliwości dostępu – ciąg dalszy*

| Nazwa możliwości dostępu | Identyfikator SID |
|---|---|
| *Software and hardware certificates or a smart card* (certyfikaty oprogramowania i sprzętu lub karta inteligentna) | S-1-15-3-9 |
| *Removable storage* (pamięć przenośna) | S-1-15-3-10 |
| *Your appointments* (Twoje spotkania) | S-1-15-3-11 |
| *Your contacts* (Twoje kontakty) | S-1-15-3-12 |
| *Internet Explorer* | S-1-15-3-4096 |

Za pomocą polecenia `Get-NtSid` możesz utworzyć zapytanie dotyczące identyfikatorów SID pakietów i możliwości dostępu (listing 4.20).

*Listing 4.20. Tworzenie listy identyfikatorów SID pakietów i możliwości dostępu*

```
PS> Get-NtSid -PackageName 'my_package' -ToSddl
S-1-15-2-4047469452-4024960472-3786564613-914846661-3775852572-3870680127 ❶
-2256146868

PS> Get-NtSid -PackageName 'my_package' -RestrictedPackageName "CHILD" -ToSddl ❷
S-1-15-2-4047469452-4024960472-3786564613-914846661-3775852572-3870680127
-2256146868-951732652-158068026-753518596-3921317197

PS> Get-NtSid -KnownSid CapabilityInternetClient -ToSddl ❸
S-1-15-3-1

PS> Get-NtSid -CapabilityName registryRead -ToSddl ❹
S-1-15-3-1024-1065365936-1281604716-3511738428-1654721687-432734479
-3232135806-4053264122-3456934681

PS> Get-NtSid -CapabilityName registryRead -CapabilityGroup -ToSddl ❺
S-1-5-32-1065365936-1281604716-3511738428-1654721687-432734479-3232135806
-4053264122-3456934681
```

Utworzono tutaj dwa identyfikatory SID pakietów i dwa identyfikatory SID możliwości dostępu. Pierwszy identyfikator SID pakietu wygenerowano przez podanie jego nazwy w poleceniu `Get-NtSid` i otrzymano wynikowy identyfikator SID ❶. Identyfikator ten utworzono na bazie nazwy w postaci małych liter, którą poddano mieszaniu za pomocą algorytmu skrótu SHA256. 256-bitowy skrót dzielony jest na siedem 32-bitowych segmentów, które pełnią rolę identyfikatorów RID. Ostatnia 32-bitowa wartość skrótu jest odrzucana.

System Windows obsługuje również ograniczony identyfikator SID pakietu mający na celu umożliwienie pakietowi utworzenia nowych bezpiecznych pakietów podrzędnych, które nie mogą wchodzić ze sobą w interakcje. Klasyczna przeglądarka internetowa Edge używała tej opcji do oddzielania obiektów potomnych mających dostęp do internetu i intranetu, tak aby w przypadku naruszenia zabezpieczeń jednego z nich nie

było możliwe uzyskanie dostępu do danych innego obiektu. Aby utworzyć obiekt potomny, użyto nazwy rodziny oryginalnego pakietu oraz identyfikatora tego obiektu ❷. Utworzony identyfikator SID rozszerza identyfikator SID oryginalnego pakietu o kolejne cztery identyfikatory RID, co można zobaczyć w danych wynikowych.

Pierwszy identyfikator SID możliwości dostępu ❸ reprezentuje starszą możliwość pozwalającą uzyskać dostęp do internetu. Zauważ, że wynikowy identyfikator SID SDDL zawiera jedną dodatkową wartość identyfikatora RID (1). Drugi identyfikator SID pochodzi od nazwy (w tym przypadku `registryRead`) ❹ używanej do umożliwienia uzyskania dostępu z uprawnieniem odczytu do grupy kluczy rejestru systemowego. Podobnie jak w przypadku identyfikatora SID pakietu nazwane identyfikatory RID możliwości dostępu są generowane na podstawie skrótu SHA256 nazwy w postaci małych liter. Aby można było odróżnić starsze identyfikatory SID możliwości dostępu od tych nazwanych, jako drugi identyfikator RID ustawiana jest wartość 1024, po której następuje skrót SHA256. Za pomocą tej metody możesz generować własne identyfikatory SID możliwości dostępu, choć metoda ta nie pozwoli na zbyt wiele, chyba że zasób skonfigurowano z myślą o jej użyciu.

System Windows obsługuje również *grupę możliwości dostępu* (ang. *capability group*), której identyfikator SID można dodać do normalnej listy grup ❺. Grupa ta ustawia pierwszy identyfikator RID o wartości 32, a w przypadku pozostałych identyfikatorów RID stosuje ten sam skrót SHA256, który został wygenerowany na podstawie nazwy możliwości dostępu.

Gdy już dostępne są identyfikatory SID, można utworzyć token *lowbox* w sposób zaprezentowany w listingu 4.21.

*Listing 4.21. Tworzenie tokena lowbox i wyszczególnienie jego właściwości*

```
PS> $token = Get-NtToken -LowBox -PackageSid 'my_package'  ❶
-CapabilitySid "registryRead", "S-1-15-3-1"
PS> Get-NtTokenGroup $token -Capabilities | Select-Object Name  ❷
Name
----
NAMED CAPABILITIES\Registry Read
APPLICATION PACKAGE AUTHORITY\Your Internet connection

PS> $package_sid = Get-NtTokenSid $token -Package -ToSddl  ❸
PS> $package_sid
S-1-15-2-4047469452-4024960472-3786564613-914846661-3775852572-3870680127
-2256146868

PS> Get-NtTokenIntegrityLevel $token
Low  ❹

PS> $token.Close()
```

Najpierw wywołano polecenie `Get-NtToken`, przekazując mu nazwę pakietu (identyfikator SID SDDL również się sprawdzi) oraz listę możliwości dostępu, które mają zostać przypisane tokenowi *lowbox* ❶. Dalej można uzyskać listę możliwości dostępu ❷. Zauważ, że różne są nazwy dwóch identyfikatorów SID możliwości dostępu: identyfikator

SID pochodzący od nazwy poprzedzono przedrostkiem `NAMED CAPABILITIES`. Nie jest możliwe przekształcenie nazwanego identyfikatora SID możliwości dostępu z powrotem w nazwę, z której on się wywodzi. Moduł narzędzia PowerShell musi generować nazwę na podstawie dużej listy znanych możliwości dostępu. Drugi identyfikator SID to starszy identyfikator SID, dlatego podsystem LSASS może z powrotem przekształcić go w nazwę.

W dalszej kolejności uzyskano identyfikator SID pakietu ❸. Ponieważ identyfikator SID pakietu uzyskano z nazwy przy użyciu algorytmu skrótu SHA256, nie jest możliwe przekształcenie go z powrotem w nazwę pakietu. I tym razem moduł narzędzia PowerShell dysponuje listą nazw, którą może wykorzystać do ustalenia, jaka była pierwotna nazwa.

Token *lowbox* zawsze ma ustawiony poziom integralności `Low` ❹. W rzeczywistości, jeśli uprzywilejowany użytkownik zmieni poziom integralności na `Medium` lub wyższy, zostaną usunięte wszystkie właściwości tokena *lowbox*, takie jak identyfikatory SID pakietów i identyfikatory SID możliwości dostępu, a token zostanie przywrócony do postaci tokena, który nie jest tokenem „piaskownicy".

Wyjaśniono, jak zmniejszyć uprawnienia użytkownika, przekształcając jego token w token „piaskownicy". Pora przenieść się na drugą stronę i przyjrzeć się temu, co sprawia, że użytkownik ma uprawnienia wystarczające do administrowania systemem Windows.

## Co czyni użytkownika administratorem?

Jeśli masz doświadczenie z zakresu systemu Unix, będziesz kojarzyć identyfikator użytkownika 0 jako reprezentujący konto administratora, czyli *root*. Dysponując tym kontem, możesz uzyskać dostęp do dowolnego zasobu i skonfigurować system według własnego uznania. Gdy instalujesz system Windows, pierwsze konfigurowane konto również będzie kontem administratora. W przeciwieństwie jednak do konta *root* konto to nie będzie miało specjalnego identyfikatora SID, który system traktuje inaczej. Co zatem decyduje w systemie Windows o tym, że konto jest kontem administratora?

Podstawowa odpowiedź jest taka, że system Windows jest tak skonfigurowany, by zapewniać specjalny dostęp określonym grupom i uprawnieniom. Dostęp na poziomie administratora jest właściwie uznaniowy, co oznacza, że możliwe jest zostanie administratorem, ale nadal z zablokowanym dostępem do zasobów. Nie ma prawdziwego odpowiednika konta *root* (choć konto użytkownika *SYSTEM* jest bliskie temu statusowi).

Administratorów opisują zazwyczaj trzy cechy. Po pierwsze, gdy konfigurujesz konto użytkownika jako administratora, dodajesz je zwykle do grupy *BUILTIN\Administrators*, a następnie konfigurujesz system Windows tak, by umożliwił dostęp do grupy podczas sprawdzania dostępu. Na przykład foldery systemowe, takie jak *C:\Windows*, są tak skonfigurowane, żeby grupa mogła tworzyć nowe pliki i katalogi.

Po drugie administratorzy mają dostęp do dodatkowych uprawnień, które skutecznie omijają elementy kontroli zabezpieczeń systemu. Na przykład uprawnienie `SeDebugPrivilege` pozwala użytkownikowi uzyskać pełny dostęp do dowolnego innego procesu lub wątku w systemie niezależnie od tego, jakie zabezpieczenia zostały im przypisane. Mając pełny dostęp do procesu, można wprowadzić do niego kod w celu uzyskania uprawnień innego użytkownika.

Po trzecie administratorzy korzystają zwykle z poziomu integralności `High`, natomiast usługi systemowe działają na poziomie `System`. Zwiększając poziom integralności administratora, utrudnia się przypadkowe pozostawienie jego zasobów (a zwłaszcza procesów i wątków) jako dostępnych dla zwykłych użytkowników. Słaba kontrola dostępu do zasobów to często wynik błędnej konfiguracji. Jeśli jednak zasób oznaczono również za pomocą poziomu integralności większego niż `Medium`, użytkownicy bez uprawnień administratora nie będą mogli dokonywać zapisu w zasobie.

Szybką metodą zweryfikowania, czy token jest tokenem administratora, jest sprawdzenie właściwości `Elevated` obiektu `Token`. Właściwość wskazuje, czy token zawiera określone grupy i dostępne uprawnienia znajdujące się na stałej liście w jądrze. Listing 4.22 prezentuje przykład dotyczący użytkownika bez uprawnień administratora.

*Listing 4.22. Właściwość Elevated w przypadku użytkownika bez uprawnień administratora*

```
PS> $token = Get-NtToken
PS> $token.Elevated
False
```

Jeśli token uwzględnia jedno z następujących uprawnień, jest automatycznie uznawany za zapewniający większe uprawnienia:

- `SeCreateTokenPrivilege`,
- `SeTcbPrivilege`,
- `SeTakeOwnershipPrivilege`,
- `SeLoadDriverPrivilege`,
- `SeBackupPrivilege`,
- `SeRestorePrivilege`,
- `SeDebugPrivilege`,
- `SeImpersonatePrivilege`,
- `SeRelabelPrivilege`,
- `SeDelegateSessionUserImpersonatePrivilege`.

Uprawnienie nie musi być włączone. Wystarczy, że jest dostępne w tokenie.

W przypadku grup ze zwiększonymi uprawnieniami jądro nie zawiera stałej listy identyfikatorów SID. Zamiast tego sprawdza tylko ostatni identyfikator RID identyfikatora SID. Jeśli dla identyfikatora RID ustawiono jedną z wartości, czyli `114`, `498`, `512`, `516`, `517`, `518`, `519`, `520`, `521`, `544`, `547`, `548`, `549`, `550`, `551`, `553`, `554`, `556` lub `569`, identyfikator SID uznawany jest za oferujący większe uprawnienia. Na przykład identyfikator SID grupy *BUILTIN\Administrators* to `S-1-4-32-544`. Ponieważ wartość `544` znajduje się na tej liście, identyfikator SID jest traktowany jako zapewniający większe uprawnienia (zauważ, że identyfikator SID `S-1-1-2-3-4-544` również byłby uznawany za dający większe uprawnienia nawet pomimo tego, że nie ma w nim nic specjalnego).

### WYSOKI POZIOM INTEGRALNOŚCI NIE JEST RÓWNOZNACZNY Z UPRAWNIENIAMI ADMINISTRATORA

Powszechnym błędnym przekonaniem jest to, że jeśli token ma poziom integralności `High`, jest to token administratora. Właściwość `Elevated` nie sprawdza jednak poziomu integralności tokena, a jedynie jego uprawnienia i grupy. Grupa *BUILTIN\Administrators* nadal działałaby z niższym poziomem integralności, umożliwiając dostęp do zasobów, takich jak katalog systemu plików systemu Windows. Jedynym ograniczeniem jest to, że niektóre uprawnienia wyższego poziomu, takie jak `SeDebugPrivilege`, nie mogą zostać włączone, jeżeli poziom integralności jest mniejszy niż `High`.

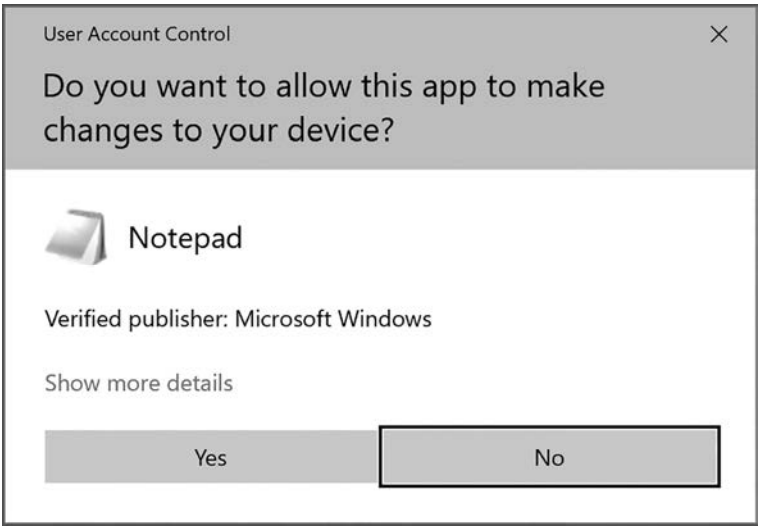
Możliwe jest również, aby użytkownik inny niż administrator korzystał z poziomu integralności `High` tak, jak to ma miejsce w przypadku procesów z dostępem do interfejsu użytkownika, które czasami działają na tym poziomie integralności, ale nie mają przyznanych żadnych specjalnych uprawnień lub grup zapewniających im status administratora.

# Kontrola konta użytkownika

Wspomniałem, że po zainstalowaniu nowej kopii systemu Windows pierwsze konto użytkownika, jakie tworzysz, to zawsze jest administrator. Ważne jest skonfigurowanie konta użytkownika w ten sposób, ponieważ inaczej nie byłoby możliwe modyfikowanie systemu ani instalowanie nowego oprogramowania.

Przed wprowadzeniem systemu Windows Vista to domyślne zachowanie stanowiło jednak ogromny problem z punktu widzenia bezpieczeństwa, ponieważ przeciętny użytkownik tworzył domyślne konto i prawdopodobnie nigdy go nie zmieniał na inne. Oznaczało to, że większość osób korzystała z pełnego konta administratora do realizowania codziennych czynności, takich jak przeglądanie internetu. Jeśli atakujący ze złymi zamiarami byłby w stanie wykorzystać lukę w zabezpieczeniach przeglądarki internetowej użytkownika, mógłby uzyskać pełną kontrolę nad komputerem z systemem Windows. W czasach przed rozpowszechnieniem się izolowania w „piaskownicy" stanowiło to poważne zagrożenie.

W systemie Windows Vista firma Microsoft zmieniła ten domyślny sposób działania, wprowadzając komponent kontroli konta użytkownika (UAC — *User Account Control*) oraz konto administratora z podzielonym tokenem. W tym modelu domyślny użytkownik pozostaje administratorem, ale z założenia wszystkie programy działają z tokenem, którego grupy administratorów i ich uprawnienia zostały usunięte. Gdy użytkownik musi wykonać zadanie administracyjne, system zwiększa uprawnienia procesu do poziomu pełnych uprawnień administratora i wyświetla okno komunikatu, takie jak widoczne na rysunku 4.3, z prośbą o potwierdzenie przez użytkownika przed kontynuowaniem.




*Rysunek 4.3. Okno dialogowe komponentu UAC z prośbą o zgodę na zwiększenie uprawnień*

Aby ułatwić użytkownikom korzystanie z systemu Windows, można tak skonfigurować program, aby wymuszał zwiększanie uprawnień przy uruchamianiu. Właściwość programu związana z podnoszeniem uprawnień jest przechowywana w pliku XML manifestu osadzonym w obrazie wykonywalnym. Wykonaj przykładowe polecenie z listingu 4.23, aby uzyskać informacje o manifeście dla wszystkich plików wykonywalnych w katalogu *System32*.

*Listing 4.23. Uzyskiwanie informacji o manifeście plików wykonywalnych*

```
PS> ls C:\Windows\System32\*.exe | Get-Win32ModuleManifest
Name                          UiAccess   AutoElevate     ExecutionLevel
----                          --------   -----------     --------------
aitstatic.exe                 False      False           asInvoker
alg.exe                       False      False           asInvoker
appidcertstorecheck.exe       False      False           asInvoker
appidpolicyconverter.exe      False      False           asInvoker
ApplicationFrameHost.exe      False      False           asInvoker
appverif.exe                  False      False           highestAvailable
--cięcie--
```

W przypadku specjalnego programu zatwierdzonego przez firmę Microsoft manifest może określać, czy automatycznie i bez potwierdzenia powinny być zwiększane uprawnienia programu (wskazuje to wartość `True` w kolumnie `AutoElevate`). Manifest wskazuje też, czy proces może działać z dostępem do interfejsu użytkownika (zagadnienie to zostanie omówione w dalszej części rozdziału). Istnieją następujące trzy możliwe wartości w kolumnie `ExecutionLevel`:

**`asInvoker`**. Powoduje uruchomienie procesu z uprawnieniami użytkownika, który go utworzył. Jest to domyślne ustawienie.

**highestAvailable**. Jeśli użytkownik jest administratorem z podzielonym tokenem, ustawienie wymusza podniesienie go do poziomu normalnego tokena administratora. W przeciwnym razie proces zostanie uruchomiony z uprawnieniami użytkownika, który go utworzył.

**requireAdministrator**. Powoduje wymuszenie zwiększenia uprawnień niezależnie od tego, czy użytkownik jest administratorem z podzielonym tokenem, czy nie. Jeżeli użytkownik nie jest administratorem, zostanie poproszony o podanie hasła konta administratora.

Gdy coś tworzy plik wykonywalny ze zwiększonym poziomem uprawnień wykonywania, powłoka wywołuje metodę RPC `RAiLaunchAdminProcess`. Sprawdza ona manifest i rozpoczyna proces podnoszenia uprawnień uwzględniający wyświetlenie okna dialogowego z prośbą o zgodę. Można również ręcznie zwiększyć uprawnienia dowolnej aplikacji za pomocą interfejsu API `ShellExecute` (zaprezentowano go w punkcie „Interfejsy API powłoki" rozdziału 3.), żądając operacji `runas`. Narzędzie PowerShell udostępnia to rozwiązanie za pośrednictwem polecenia `Start-Process` stosowanego w następujący sposób:

```
PS> Start-Process notepad -Verb runas
```

Po wykonaniu tego polecenia powinieneś ujrzeć okno dialogowe komponentu UAC z prośbą o zgodę. Jeśli klikniesz w nim przycisk *Tak*, plik *notepad.exe* powinien zostać uruchomiony na pulpicie z uprawnieniami administratora.

## Tokeny powiązane i typ zwiększania uprawnień

Gdy administrator uwierzytelnia się na pulpicie, system śledzi następujące dwa tokeny użytkownika:

**Limitowany**. Token bez zwiększonych uprawnień stosowany przez większość działających procesów.

**Pełny**. Pełny token administratora używany tylko po zwiększeniu uprawnień.

Określenie *administrator z podzielonym tokenem* (ang. *split-token administrator*) ma związek z tymi dwoma tokenami, ponieważ przyznawany użytkownikowi dostęp jest dzielony pomiędzy token limitowany i pełny.

Obiekt `Token` zawiera pole służące do powiązania ze sobą tych dwóch tokenów. Token powiązany można uzyskać za pomocą wywołania systemowego `NtQueryInformationToken` i klasy informacyjnej `TokenLinkedToken`. W listingu 4.24 za pomocą narzędzia PowerShell sprawdzono niektóre właściwości tych tokenów powiązanych.

*Listing 4.24. Wyświetlenie właściwości tokena powiązanego*

```
PS> Use-NtObject($token = Get-NtToken -Linked) {  ❶
    Format-NtToken $token -Group -Privilege -Integrity -Information
}
GROUP SID INFORMATION
-----------------
```

```
Name                           Attributes
----                           ----------
BUILTIN\Administrators         Mandatory, EnabledByDefault, Enabled, Owner  ❷
--cięcie--
PRIVILEGE INFORMATION
---------------------
Name                           Luid               Enabled
----                           ----               -------
SeIncreaseQuotaPrivilege       00000000-00000005  False
SeSecurityPrivilege            00000000-00000008  False  ❸
SeTakeOwnershipPrivilege       00000000-00000009  False
--cięcie--

INTEGRITY LEVEL
---------------
High  ❹

TOKEN INFORMATION
-----------------
Type           : Impersonation  ❺
Imp Level      : Identification
Auth ID        : 00000000-0009361F
Elevated       : True  ❻
Elevation Type: Full  ❼
Flags          : NotLow
```

Przekazując parametr `Linked` poleceniu `Get-NtToken` ❶, uzyskano dostęp do tokena powiązanego, a następnie sformatowano go w celu wyświetlenia jego grup, uprawnień i poziomu integralności oraz informacji o nim. Na liście grup można zobaczyć, że grupa *BUILTIN\Administrators* jest włączona ❷. Możliwe jest również stwierdzenie, że lista uprawnień zawiera niektóre uprawnienia wyższego poziomu, takie jak `SeSecurityPrivilege` ❸. Kombinacja grup i uprawnień potwierdza, że jest to token administratora.

Dla tokena ustawiono poziom integralności `High` ❹, co jak wspomniano wcześniej, zapobiega przypadkowemu pozostawieniu poufnych zasobów jako dostępnych dla użytkowników innych niż administrator. W informacjach o tokenie widać, że istnieje token personifikacji z poziomem `Identification` ❺. Aby uzyskać token, który może utworzyć nowy proces, obiekt wywołujący wymaga uprawnienia `SeTcbPrivilege`. Oznacza ono, że token mogą uzyskać tylko usługi systemowe, takie jak usługa informacji o aplikacji. Na końcu widać, że token oznaczono jako mający zwiększone uprawnienia ❻, a ponadto że typ podniesienia uprawnień tokena wskazuje, że jest to token pełny ❼. Porównajmy to z tokenem limitowanym (listing 4.25).

*Listing 4.25. Wyświetlenie właściwości tokena limitowanego*

```
PS> Use-NtObject($token = Get-NtToken) {  ❶
    Format-NtToken $token -Group -Privilege -Integrity -Information
}
GROUP SID INFORMATION
-----------------
Name                           Attributes
----                           ----------
```

```
BUILTIN\Administrators UseForDenyOnly  ❷
--cięcie--

PRIVILEGE INFORMATION
---------------------
Name                           Luid              Enabled
----                           ----              -------
SeShutdownPrivilege            00000000-00000013 False  ❸
SeChangeNotifyPrivilege        00000000-00000017 True
SeUndockPrivilege              00000000-00000019 False
SeIncreaseWorkingSetPrivilege 00000000-00000021 False
SeTimeZonePrivilege            00000000-00000022 False

INTEGRITY LEVEL
---------------
Medium  ❹

TOKEN INFORMATION
-----------------
Type           : Primary
Auth ID        : 00000000-0009369B
Elevated       : False  ❺
Elevation Type: Limited  ❻
Flags          : VirtualizeAllowed, IsFiltered, NotLow  ❼
```

Najpierw uzyskano uchwyt do bieżącego tokena i sformatowano go w taki sam sposób, jak w listingu 4.24 ❶. Na liście grup widać, że grupa `BUILTIN\Administrators` została przekształcona w grupę `UseForDenyOnly` ❷. Dowolna inna grupa, która będzie spełniać wymogi sprawdzenia identyfikatora RID zwiększonych uprawnień, zostanie przekształcona w identyczny sposób.

Lista uprawnień zawiera tylko pięć pozycji ❸. Jest to jedyne pięć uprawnień, jakie może mieć token limitowany. Dla tokena ustawiono poziom integralności `Medium`, czyli mniejszy od poziomu `High` tokena pełnego ❹. W informacjach o tokenie widać, że nie ma on zwiększonych uprawnień ❺, a typ podnoszenia uprawnień wskazuje, że jest to token limitowany ❻.

Na koniec zauważ, że flagi zawierają wartość `IsFiltered` ❼. Wskazuje ona, że token został poddany filtrowaniu za pomocą wywołania systemowego `NtFilterToken`. Wynika to stąd, że w celu utworzenia tokena limitowanego podsystem LSASS utworzy najpierw nowy token pełny, aby jego identyfikator uwierzytelnienia miał unikalną wartość (jeśli porównasz wartości typu `Auth ID` w listingach 4.24 i 4.25, zauważysz, że są one rzeczywiście inne). Dzięki temu monitor SRM traktuje te dwa tokeny jako należące do oddzielnych sesji logowania. Podsystem LSASS przekazuje następnie token wywołaniu `NtFilterToken` z flagą parametru `LuaToken`, aby przekształcić dowolną grupę ze zwiększonymi uprawnieniami w oznaczoną flagą `UseForDenyOnly` i usunąć wszystkie uprawnienia oprócz pięciu dozwolonych. Wywołanie `NtFilterToken` nie obniża jednak poziomu integralności z `High` na `Medium`. Musi to zostać wykonane osobno. Na koniec podsystem LSASS stosuje wywołanie `NtSetInformationToken` w celu powiązania ze sobą dwóch tokenów za pomocą klasy informacyjnej `TokenLinkedToken`.

Istnieje trzeci *domyślny* typ podnoszenia uprawnień używany dla każdego tokena, który nie jest skojarzony z kontem administratora z podzielonym tokenem:

```
PS> Use-NtObject($token = Get-NtToken -Anonymous) { $token.ElevationType }
Default
```

W tym przykładzie anonimowy użytkownik nie jest administratorem korzystającym z podzielonego tokena, dlatego token ma domyślny typ podnoszenia uprawnień.

## Dostęp do interfejsu użytkownika

Jedną z innych opcji zabezpieczeń wprowadzonych w systemie Windows Vista jest izolowanie uprawnień UIPI (*User Interface Privilege Isolation*), które uniemożliwia procesowi o niższych uprawnieniach interakcję w sposób programowy z interfejsem użytkownika procesu o wyższych uprawnieniach. Jest to wymuszane za pomocą poziomów integralności i stanowi kolejny powód, dla którego administratorzy komponentu UAC korzystają z poziomu integralności `High`.

Izolowanie UIPI stanowi jednak problem w przypadku aplikacji, które zaprojektowano pod kątem interakcji z interfejsem użytkownika, takich jak czytniki ekranowe i klawiatury dotykowe. Aby obejść to ograniczenie bez przyznawania procesowi zbyt dużych uprawnień, token może ustawić flagę dostępu do interfejsu użytkownika. To, czy proces uzyska dostęp do interfejsu użytkownika, zależy od ustawienia `UiAccess` w pliku manifestu pliku wykonywalnego.

Ta flaga dostępu do interfejsu użytkownika sygnalizuje środowisku pulpitu, że powinno ono wyłączyć sprawdzanie izolowania UIPI. W listingu 4.26 uzyskano informacje o fladze w odpowiednim procesie klawiatury ekranowej OSK (*On-Screen Keyboard*).

*Listing 4.26. Uzyskiwanie informacji o fladze dostępu do interfejsu użytkownika w tokenie podstawowym procesu klawiatury ekranowej*

```
PS> $process = Start-Process "osk.exe" -PassThru
PS> $token = Get-NtToken -ProcessId $process.Id
PS> $token.UIAccess
True
```

Aby sprawdzić flagę dostępu do interfejsu użytkownika, uruchomiono proces klawiatury OSK i otwarto jego obiekt `Token`. W celu ustawienia tej flagi obiekt wywołujący musi posiadać uprawnienie `SeTcbPrivilege`. Jedynym sposobem na utworzenie procesu z dostępem do interfejsu użytkownika jako zwykły użytkownik jest użycie komponentu UAC. W związku z tym każdy proces z dostępem do interfejsu użytkownika musi zostać uruchomiony za pomocą wywołania `ShellExecute`. Z tego powodu w listingu 4.26 zastosowano polecenie `Start-Process`. Wszystko to dzieje się w tle podczas tworzenia aplikacji z dostępem do interfejsu użytkownika.

## Wirtualizacja

Kolejny problem, który pojawił się w systemie Vista z powodu komponentu UAC, dotyczy tego, jak obsługiwać starsze aplikacje, które oczekują możliwości zapisu w lokalizacjach dostępnych tylko dla administratorów, takich jak katalog *Windows* lub gałąź rejestru komputera lokalnego. W systemie Windows Vista zaimplementowano specjalne obejście: jeśli w tokenie podstawowym włączono flagę wirtualizacji, będzie on automatycznie przekierowywać operacje zapisu z tych lokalizacji do magazynu poszczególnych użytkowników. Sprawiało to, że proces „uznawał", że pomyślnie dodał zasoby do chronionych lokalizacji.

Domyślnie flaga wirtualizacji jest automatycznie włączana w przypadku starszych aplikacji. Można ją jednak określić ręcznie, ustawiając odpowiednią właściwość w tokenie podstawowym. W powłoce bez uprawnień administratora uruchom polecenia z listingu 4.27.

*Listing 4.27. Włączanie wirtualizacji w obrębie obiektu Token i tworzenie pliku w katalogu C:\Windows*

```
PS> $file = New-NtFile -Win32Path C:\Windows\witaj.txt -Access GenericWrite  ❶
New-NtFile : (0xC0000022) - {Access Denied}
A process has requested access to an object, but has not been granted those
access rights.

PS> $token = Get-NtToken
PS> $token.VirtualizationEnabled = $true  ❷
PS> $file = New-NtFile -Win32Path C:\Windows\witaj.txt -Access GenericWrite  ❸
PS> $file.Win32PathName  ❹
C:\Users\user\AppData\Local\VirtualStore\Windows\witaj.txt
```

W tym listingu najpierw podjęto próbę utworzenia pliku `C:\Windows\witaj.txt` umożliwiającego zapis ❶. Operacja ta kończy się niepowodzeniem z wygenerowanym wyjątkiem odmowy dostępu. Dalej uzyskano bieżący token podstawowy i dla właściwości `VirtualizationEnabled` ustawiono wartość `True` ❷. Powtórna operacja tworzenia pliku zakończyła się powodzeniem ❸. Jeśli utworzy się zapytanie dotyczące lokalizacji pliku, okazuje się, że znajduje się on w katalogu użytkownika wewnątrz wirtualnego magazynu ❹. Tylko zwykłe, nieuprzywilejowane tokeny mogą aktywować wirtualizację. Tokeny usług systemowych i administratorów mają wirtualizację wyłączoną. Aby dowiedzieć się, czy wirtualizacja jest dozwolona, utwórz zapytanie dotyczące właściwości `VirtualizationAllowed` obiektu `Token`.

# Atrybuty zabezpieczeń

*Atrybuty zabezpieczeń* tokena to lista par złożonych z nazwy i wartości, które zapewniają dowolne dane. Istnieją trzy typy atrybutów zabezpieczeń powiązanych z tokenem: *lokalne* (ang. *local*), *oświadczenia dotyczące użytkownika* (ang. *user claims*) i *oświadczenia dotyczące urządzenia* (ang. *device claims*). Każdy atrybut zabezpieczeń może mieć jedną lub więcej wartości, które muszą być tego samego typu. W tabeli 4.5 zebrano poprawne typy atrybutu zabezpieczeń.

*Tabela 4.5. Typy atrybutów zabezpieczeń*

| Nazwa typu | Opis |
|---|---|
| `Int64` | 64-bitowa liczba całkowita ze znakiem. |
| `UInt64` | 64-bitowa liczba całkowita bez znaku. |
| `String` | Ciąg znaków Unicode. |
| `Fqbn` | W pełni kwalifikowana nazwa binarna zawierająca numer wersji i ciąg znaków Unicode. |
| `Sid` | Identyfikator SID. |
| `Boolean` | Wartość prawda lub fałsz przechowywana jako typ `Int64`, gdzie 0 oznacza fałsz, a 1 oznacza prawdę. |
| `OctetString` | Dowolna tablica bajtów. |

Zestaw flag może zostać przypisany do atrybutu zabezpieczeń w celu zmiany aspektów jego działania (określających na przykład, czy nowe tokeny mogą go dziedziczyć). W tabeli 4.6 zamieszczono zdefiniowane flagi.

*Tabela 4.6. Flagi atrybutów zabezpieczeń*

| Nazwa flagi | Opis |
|---|---|
| `NonInheritable` | Atrybut zabezpieczeń nie może być dziedziczony przez token procesu potomnego. |
| `CaseSensitive` | Jeśli atrybut zabezpieczeń zawiera wartość łańcuchową, porównanie powinno uwzględniać wielkość liter. |
| `UseForDenyOnly` | Atrybut zabezpieczeń jest używany tylko podczas sprawdzania pod kątem odmowy dostępu. |
| `DisabledByDefault` | Atrybut zabezpieczeń jest domyślnie wyłączony. |
| `Disabled` | Atrybut zabezpieczeń jest wyłączony. |
| `Mandatory` | Atrybut zabezpieczeń jest obowiązkowy. |
| `Unique` | Atrybut zabezpieczeń powinien być unikalny w systemie lokalnym. |
| `InheritOnce` | Atrybut zabezpieczeń może być dziedziczony raz przez proces potomny, a następnie powinien mieć ustawioną flagę `NonInheritable`. |

Prawie każdy token procesu ma atrybut zabezpieczeń `TSA://ProcUnique`. Atrybut ten zawiera unikalny identyfikator LUID przypisany podczas tworzenia procesu. Używając polecenia `Show-NtTokenEffective`, można wyświetlić wartość atrybutu dla efektywnego tokena (listing 4.28).

*Listing 4.28. Uzyskiwanie atrybutów zabezpieczeń dla bieżącego procesu*

```
PS> Show-NtTokenEffective -SecurityAttributes
SECURITY ATTRIBUTES
-------------------
Name             Flags                  ValueType Values
----             -----                  --------- ------
TSA://ProcUnique NonInheritable, Unique UInt64    {133, 1592482}
```

W danych wynikowych widać, że nazwa atrybutu to `TSA://ProcUnique`. Ma on dwie wartości typu `UInt64`, które połączone tworzą identyfikator LUID. Atrybut zawiera też dwie flagi: `NonInheritable` (oznacza, że atrybut zabezpieczeń nie będzie przekazywany nowym tokenom procesu) oraz `Unique` (powoduje, że jądro nie powinno próbować scalać atrybutu zabezpieczeń z żadnym innym atrybutem w systemie o tej samej nazwie).

Aby ustawić lokalne atrybuty zabezpieczeń, przed użyciem wywołania `NtSetInformationToken` obiekt wywołujący musi dysponować uprawnieniem `SeTcbPrivilege`. Oświadczenia dotyczące użytkownika i urządzenia muszą zostać ustawione podczas tworzenia tokena, co omówiono w następnym podrozdziale.

# Tworzenie tokenów

Podsystem LSASS tworzy zwykle tokeny, gdy użytkownik uwierzytelnia się na komputerze. Może też jednak tworzyć tokeny dla kont użytkowników, które nie istnieją, takich jak konta wirtualne używane dla usług. Tokeny te mogą być interaktywne i przewidziane do zastosowania w sesji konsoli albo mogą być tokenami sieciowymi wykorzystywanymi w sieci lokalnej. Wywołując interfejs API podsystemu Win32, taki jak `LogonUser`, który odwołuje się do podsystemu LSASS w celu utworzenia tokena, lokalnie uwierzytelniony użytkownik może utworzyć token innego użytkownika.

Aż do rozdziału 10. podsystem LSASS nie będzie szczegółowo omawiany. Warto jednak zrozumieć, w jaki sposób tworzy on tokeny. Aby to zrealizować, podsystem stosuje wywołanie systemowe `NtCreateToken`. Jak wcześniej wspomniano, wywołanie wymaga uprawnienia `SeCreateTokenPrivilege`, które jest przyznawane ograniczonej liczbie procesów. Jest to jedno z najbardziej uprzywilejowanych uprawnień, ponieważ pozwala na tworzenie dowolnych tokenów w przypadku identyfikatora SID dowolnej grupy lub użytkownika oraz uzyskiwanie dostępu do każdego zasobu na komputerze lokalnym.

Choć rzadko konieczne będzie stosowanie wywołania `NtCreateToken` z poziomu narzędzia PowerShell, można to zrobić za pośrednictwem polecenia `New-NtToken`, pod warunkiem że włączono uprawnienie `SeCreateTokenPrivilege`. Wywołanie systemowe `NtCreateToken` akceptuje następujące parametry:

**Typ tokena** — token podstawowy lub token personifikacji.

**Identyfikator uwierzytelnienia** — identyfikator uwierzytelnienia LUID; można ustawić dla niego dowolną wartość.

**Czas wygaśnięcia** — umożliwia wygaśnięcie tokena po określonym czasie.

**Użytkownik** — identyfikator SID użytkownika.

**Grupy** — lista identyfikatorów SID grup.

**Uprawnienia** — lista uprawnień.

**Właściciel** — identyfikator SID właściciela.

**Grupa podstawowa** — identyfikator SID grupy podstawowej.

**Źródło** — nazwa informacyjna źródła.

Ponadto w systemie Windows 8 wprowadzono w wywołaniu systemowym następujące nowe opcje, z których można skorzystać za pośrednictwem wywołania systemowego `NtCreateTokenEx`:

**Grupy urządzeń** — lista dodatkowych identyfikatorów SID dla urządzenia.

**Atrybuty oświadczeń dotyczących urządzenia** — lista atrybutów zabezpieczeń definiujących oświadczenia dotyczące urządzenia.

**Atrybuty oświadczeń dotyczących użytkownika** — lista atrybutów zabezpieczeń służących do definiowania oświadczeń dotyczących użytkownika.

**Zasada obowiązkowa** — zestaw flag wskazujących obowiązkową zasadę integralności tokena.

Wszystko, co nie znajduje się na tych dwóch listach, można skonfigurować tylko przez zastosowanie wywołania `NtSetInformationToken` po utworzeniu nowego tokena. W zależności od tego, jaką ustawia się właściwość tokena, może być wymagane inne uprawnienie, takie jak `SeTcbPrivilege`. Zademonstrujemy, jak utworzyć nowy token za pomocą skryptu z listingu 4.29, który trzeba uruchomić jako administrator.

*Listing 4.29. Tworzenie nowego tokena*

```
PS> Enable-NtTokenPrivilege SeDebugPrivilege
      PS> $imp = Use-NtObject($p = Get-NtProcess -Name lsass.exe) {  ❶
    Get-NtToken -Process $p -Duplicate
}
PS> Enable-NtTokenPrivilege SeCreateTokenPrivilege -Token $imp  ❷
PS> $token = Invoke-NtToken $imp {  ❸
    New-NtToken -User "S-1-0-0" -Group "S-1-1-0"
}
PS> Format-NtToken $token -User -Group
USER INFORMATION
----------------
Name     Sid
----     ---
NULL SID S-1-0-0  ❹

GROUP SID INFORMATION
-----------------
Name                                Attributes
----                                ----------
Everyone                            Mandatory, EnabledByDefault, Enabled  ❺
Mandatory Label\System Mandatory Level Integrity, IntegrityEnabled
```

Zwykły administrator nie ma domyślnie uprawnienia `SeCreateTokenPrivilege`. Z tego powodu niezbędne jest pożyczenie tokena z innego procesu, który go zawiera. W większości przypadków najłatwiej pożyczyć token z procesu podsystemu LSASS. Otwarto ten proces oraz jego token, duplikując go do tokena personifikacji ❶. Upewniono się następnie, że uprawnienie `SeCreateTokenPrivilege` jest włączone w tokenie ❷. W dalszej kolejności można podszyć się pod token i wywołać polecenie `New-NtToken`, przekazując mu identyfikator SID konta użytkownika oraz pojedynczej grupy ❸. Na koniec można wyświetlić szczegóły dotyczące nowego tokena, w tym zestaw identyfikatorów SID jego użytkownika ❹ oraz zestaw grup ❺. Polecenie `New-NtToken` dodaje także domyślny identyfikator SID poziomu integralności systemu, który można zobaczyć na liście grup.

# Przypisywanie tokena

Jeśli zwykłe konto użytkownika mogłoby przypisywać dowolne tokeny podstawowe lub personifikacji, miałoby możliwość zwiększenia swoich uprawnień w celu uzyskania dostępu do zasobów innych użytkowników. Byłoby to szczególnie problematyczne w przypadku podszywania się, ponieważ inne konto użytkownika musiałoby jedynie otworzyć nazwany potok, aby nieumyślnie umożliwić serwerowi uzyskanie tokena personifikacji.

Z tego powodu monitor SRM nakłada ograniczenia dotyczące tego, co może zrobić zwykły użytkownik bez uprawnień `SeAssignPrimaryTokenPrivilege` i `SeImpersonationPrivilege`. Przyjrzyjmy się kryteriom, które muszą zostać spełnione, aby zwykłemu kontu użytkownika przypisać token.

## Przypisywanie tokena podstawowego

Nowemu procesowi może zostać przypisany token podstawowy na jeden z trzech następujących sposobów:

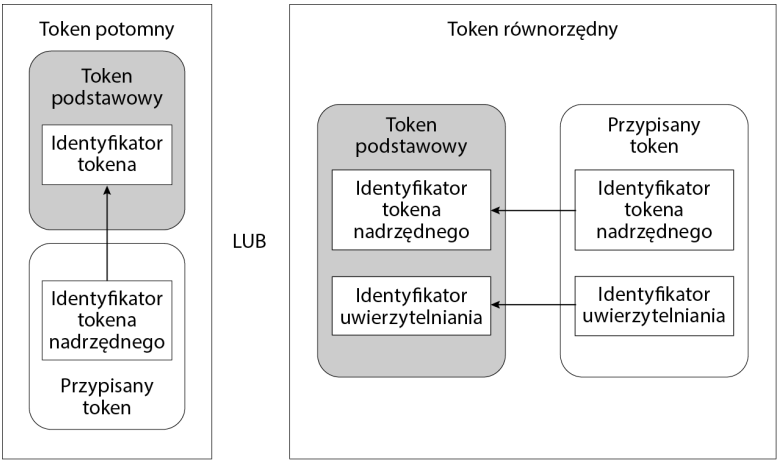
- Proces może dziedziczyć token z procesu nadrzędnego.
- Token może zostać przypisany podczas tworzenia procesu (na przykład za pomocą interfejsu API `CreateProcessAsUser`).
- Token może zostać ustawiony po utworzeniu procesu i przed uruchomieniem go przy użyciu wywołania `NtSetInformationProcess`.

Dziedziczenie tokena z procesu nadrzędnego to zdecydowanie najczęstszy sposób przypisywania tokena. Gdy na przykład uruchomiasz aplikację z menu *Start* systemu Windows, nowy proces odziedziczy token z procesu programu Explorer.

Jeśli proces nie dziedziczy tokena ze swojego procesu nadrzędnego, zostanie mu przekazany token jako uchwyt, który musi mieć prawo dostępu `AssignPrimary`. Jeżeli przyznano dostęp do obiektu `Token`, monitor SRM nakłada dodatkowe kryteria na token, aby zapobiec przypisaniu bardziej uprzywilejowanego tokena (chyba że token podstawowy obiektu wywołującego ma włączone uprawnienie `SeAssignPrimaryTokenPrivilege`).

Funkcja jądra `SeIsTokenAssignableToProcess` stosuje kryteria dotyczące tokena. Sprawdza ona najpierw, czy przypisany token ma poziom integralności taki sam jak token

podstawowy bieżącego procesu lub niższy. Jeśli to kryterium jest spełnione, funkcja sprawdza następnie, czy token spełnia jedno z kryteriów zaprezentowanych na rysunku 4.4, a mianowicie czy token jest elementem potomnym tokena podstawowego obiektu wywołującego, czy elementem równorzędnym tokena podstawowego.




*Rysunek 4.4. Kryteria przypisywania tokena podstawowego przez funkcję SeIsTokenAssignableToProcess*

Omówmy najpierw przypadek tokena potomnego. Proces użytkownika może utworzyć nowy token na podstawie istniejącego. Gdy to nastąpi, dla właściwości `ParentTokenId` obiektu jądra w nowym tokenie ustawiany jest identyfikator tokena nadrzędnego. Jeśli wartość tej właściwości nowego tokena jest zgodna z wartością identyfikatora bieżącego tokena podstawowego, przypisanie jest uznawane. Przykładem tokena potomnego jest token ograniczony. Gdy tworzysz taki token za pomocą wywołania `NtFilterToken`, jako identyfikator tokena nadrzędnego nowego tokena ustawiany jest identyfikator oryginalnego tokena.

*Token równorzędny* (ang. *sibling token*) to token tworzony jako część tej samej sesji uwierzytelniania co istniejący token. Aby sprawdzić to kryterium, funkcja porównuje identyfikator tokena nadrzędnego oraz identyfikatory uwierzytelniania dwóch tokenów. Jeżeli są one identyczne, token może zostać przypisany. Sprawdzenie to potwierdza, że sesje uwierzytelniania są specjalnymi sesjami równorzędnymi ustawionymi przez jądro (rzadka konfiguracja). Typowe przykłady tokena równorzędnego obejmują tokeny zduplikowane z tokena bieżącego procesu oraz tokeny *lowbox*.

Zauważ, że funkcja nie sprawdza użytkownika reprezentowanego przez token, a jeśli token spełnia jedno z kryteriów, możliwe jest przypisanie go do nowego procesu. Jeśli jednak nie spełnia kryteriów, podczas przypisywania tokena zostanie zwrócony błąd `STATUS_PRIVILEGE_NOT_HELD`.

Jak mimo tych ograniczeń narzędzie `runas` tworzy nowy proces jako zwykły użytkownik? Korzysta ono z interfejsu API `CreateProcessWithLogon`, który w celu ominięcia tych sprawdzeń uwierzytelnia użytkownika i uruchamia proces z poziomu usługi systemowej, która ma wymagane uprawnienia.

Próba przypisania tokenu procesu pokaże, jak łatwo operacja może się nie udać nawet wtedy, gdy tokeny przypisuje się dla tego samego użytkownika. Uruchom kod z listingu 4.30 jako użytkownik bez uprawnień administratora.

*Listing 4.30. Tworzenie procesu za pomocą tokenów ograniczonych*

```
PS> $token = Get-NtToken -Filtered -Flags DisableMaxPrivileges
PS> Use-NtObject($proc = New-Win32Process notepad -Token $token) {  ❶
    $proc | Out-Host
}
Process            : notepad.exe
Thread             : thread:11236 - process:9572
Pid                : 9572
Tid                : 11236
TerminateOnDispose : False
ExitStatus         : 259
ExitNtStatus       : STATUS_PENDING

PS> $token = Get-NtToken -Filtered -Flags DisableMaxPrivileges -Token $token  ❷
PS> $proc = New-Win32Process notepad -Token $token
Exception calling "CreateProcess" with "1" argument(s): "A required privilege  ❸
is not held by the client"
```

Utworzono tutaj dwa tokeny ograniczone i użyto ich do utworzenia instancji programu Notatnik. W ramach pierwszej próby tworzony jest token na podstawie bieżącego tokena podstawowego ❶. W polu identyfikatora tokena nadrzędnego w nowym tokenie ustawiany jest identyfikator tokena podstawowego, a gdy token zostanie użyty podczas tworzenia procesu, operacja zakończy się powodzeniem.

W drugiej próbie utworzono kolejny token, który jednak bazuje na wcześniej utworzonym tokenie ❷. Nie powiodło się tworzenie procesu z użyciem tego tokena z wygenerowanym błędem dotyczącym uprawnienia ❸. Wynika to stąd, że jako identyfikator tokena nadrzędnego drugiego tokena ustawiono identyfikator utworzonego tokena, a nie tokena podstawowego. Ponieważ token nie spełnia kryterium tokena potomnego ani tokena równorzędnego, operacja ta zakończy się niepowodzeniem podczas przypisywania.

Token można ustawić po utworzeniu procesu za pomocą wywołania systemowego `NtSetInformationProcess` lub `ProcessAccessToken`, które narzędzie PowerShell udostępnia w ramach polecenia `Set-NtToken` (demonstruje to listing 4.31).

*Listing 4.31. Ustawianie tokena zapewniającego dostęp po uruchomieniu procesu*

```
PS> $proc = Get-NtProcess -Current
PS> $token = Get-NtToken -Duplicate -TokenType Primary
PS> Set-NtToken -Process $proc -Token $token
Set-NtToken : (0xC00000BB) - The request is not supported.
```

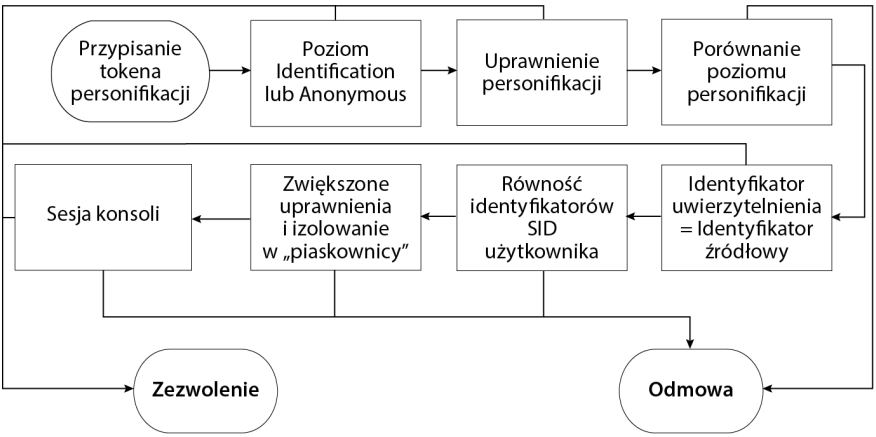
Operacja przypisywania nie pomija żadnych z wcześniej omówionych sprawdzeń dotyczących przypisywania. Gdy początkowy wątek procesu rozpoczyna działanie, wyłączona jest opcja ustawiająca token podstawowy, dlatego przy próbie ustawienia tokena dla uruchomionego procesu otrzymuje się błąd `STATUS_UNSUPPORTED`.

## Przypisywanie tokena personifikacji

Podobnie jak jest w przypadku tokenów podstawowych, monitor SRM wymaga, aby przypisany token personifikacji spełniał określony zestaw kryteriów. W przeciwnym razie odrzuci on przypisanie tokena do wątku. Co ciekawe, kryteria te różnią się od tych obowiązujących przy przypisywaniu tokenów podstawowych. Może to prowadzić do sytuacji, w których można przypisać token personifikacji, lecz nie token podstawowy, i odwrotnie.

Jeśli token określono jawnie, uchwyt musi mieć prawo dostępu `Impersonate`. Jeżeli podszywanie się (personifikacja) odbywa się niejawnie, jądro już utrzymuje token i nie wymaga żadnego konkretnego prawa dostępu.

Funkcja `SeTokenCanImpersonate` w jądrze obsługuje sprawdzenie kryteriów personifikacji. Jak widać na rysunku 4.5, sprawdzenie to jest znacznie bardziej złożone niż w przypadku przypisywania tokenów podstawowych.




*Rysunek 4.5. Wykonywane przez funkcję SeTokenCanImpersonate sprawdzenia dotyczące tokena personifikacji*

Przeanalizujmy każde sprawdzenie i opiszmy, co w ramach niego jest brane pod uwagę zarówno w przypadku tokena personifikacji, jak i tokena podstawowego. Zauważ, że ponieważ możliwe jest przypisanie tokena personifikacji do wątku w innym procesie (jeśli dysponujesz odpowiednim uchwytem tego wątku), sprawdzany token podstawowy to ten przypisany do procesu zawierającego wątek, a nie token podstawowy wątku wywołującego. Funkcja wykonuje następujące kroki weryfikacji:

1. Sprawdzenie pod kątem poziomu `Identification` lub `Anonymous`. Jeśli token personifikacji uwzględnia jeden z tych poziomów, przypisanie go do wątku nie stanowi zagrożenia z punktu widzenia bezpieczeństwa, a monitor SRM natychmiast zezwala na przypisanie. Sprawdzenie to umożliwia również przypisanie, jeżeli token personifikacji reprezentuje użytkownika anonimowego na podstawie jego identyfikatora uwierzytelnienia.
2. Sprawdzenie istnienia uprawnienia personifikacji. Jeśli włączono uprawnienie `SeImpersonatePrivilege`, monitor SRM ponownie od razu zezwala na przypisanie.

3. Porównanie poziomów integralności tokena podstawowego i tokena personifikacji. Jeśli poziom integralności pierwszego tokena jest niższy niż drugiego, przypisanie nie zostanie umożliwione. Jeżeli poziomy są takie same lub wyższe, sprawdzenia są kontynuowane.
4. Sprawdzenie, czy identyfikator uwierzytelnienia jest taki sam jak identyfikator źródłowy. Jeśli źródłowy identyfikator logowania tokena personifikacji jest identyczny jak identyfikator uwierzytelnienia tokena podstawowego, monitor SRM zezwala na przypisanie. W przeciwnym razie kontynuuje sprawdzenia.

   Zauważ, że sprawdzenie to ma interesujące konsekwencje. Jak wspomniano wcześniej w rozdziale, źródłowy identyfikator logowania tokenów zwykłych użytkowników jest ustawiony jako identyfikator uwierzytelnienia konta użytkownika *SYSTEM*. Wynika to stąd, że proces uwierzytelniania działa z użyciem tego konta. Oznacza to, że konto *SYSTEM* może podszywać się pod każdy inny token w systemie, jeśli spełnia wymóg poziomu integralności (nawet wtedy, gdy nie włączono uprawnienia `SeImpersonatePrivilege`).
5. Sprawdzenie, czy identyfikatory SID użytkownika są jednakowe. Jeśli identyfikator SID użytkownika tokena podstawowego nie jest tożsamy z identyfikatorem SID użytkownika tokena personifikacji, monitor SRM odmawia przypisania. W przeciwnym razie kontynuuje sprawdzanie. Kryterium to pozwala użytkownikowi podszywać się pod swoje własne konto, ale blokuje możliwość podszywania się pod konto innego użytkownika, chyba że użytkownik dysponuje danymi uwierzytelniającymi tego konta. Podczas uwierzytelniania innego użytkownika podsystem LSASS zwraca token personifikacji ze źródłowym identyfikatorem logowania ustawionym jako identyfikator uwierzytelnienia obiektu wywołującego. Oznacza to, że token pomyślnie przejdzie poprzednie sprawdzenie, a identyfikatory SID użytkownika nigdy nie będą porównywane.
6. Sprawdzenie pod kątem flagi `Elevated`. Sprawdzenie to zapewnia, że obiekt wywołujący nie może podszywać się pod bardziej uprzywilejowany token tego samego użytkownika. Jeśli token personifikacji ma ustawioną flagę `Elevated`, lecz token podstawowy nie, operacja personifikacji zostanie odrzucona. W wersjach systemu Windows starszych niż wersja 10 sprawdzenie to nie było przeprowadzane, dlatego wcześniej możliwe było podszywanie się pod token administratora komponentu UAC, jeśli najpierw obniżono poziom integralności.
7. Sprawdzenie pod kątem izolowania w „piaskownicy”. Działanie to zapewnia, że obiekt wywołujący nie może podszywać się pod token w mniejszym stopniu izolowany w „piaskownicy”. Aby podszywać się pod token *lowbox*, nowy token musi być zgodny z identyfikatorem SID pakietu albo musi być ograniczonym identyfikatorem SID pakietu tokena podstawowego. W przeciwnym razie personifikacja zostanie odrzucona. Żadne sprawdzenie nie jest przeprowadzane względem listy możliwości. W przypadku tokena ograniczonego wystarczy, że nowy token również jest ograniczony, jeśli nawet inna jest lista ograniczonych identyfikatorów SID. To samo dotyczy tokenów z ograniczeniem zapisu. Monitor SRM zawiera różne mechanizmy zabezpieczeń, które utrudniają uzyskanie bardziej uprzywilejowanego tokena „piaskownicy”.

8. Sprawdzenie sesji konsoli. W tym ostatnim kroku sprawdza się, czy sesja konsoli jest sesją 0, czy nie. Zapobiega to podszyciu się przez użytkownika pod token w sesji 0, która może zapewnić zwiększone uprawnienia (np. możliwość tworzenia globalnych obiektów `Section`).

Możesz uważać, że jeśli funkcja odrzuci przypisanie, zwróci błąd `STATUS_PRIVILEGE_NOT_HELD`. Tak jednak nie jest. Zamiast tego monitor SRM duplikuje token personifikacji jako token na poziomie identyfikacji i przypisuje go. Oznacza to, że jeśli nawet nie powiedzie się operacja przypisania w przypadku personifikacji, wątek nadal może sprawdzić właściwości tokena.

Korzystając z polecenia `Test-NtTokenImpersonation` narzędzia PowerShell, możesz sprawdzić, czy można się podszyć pod token. Polecenie podszywa się pod token i ponownie otwiera go z poziomu wątku. W dalszej kolejności polecenie porównuje poziom personifikacji oryginalnego tokena i ponownie otwartego tokena, po czym zwraca wynik w postaci wartości boolowskiej. W listingu 4.32 zaprezentowano prosty przykład, który zakończy się niepowodzeniem z powodu sprawdzenia poziomu integralności. Zauważ, że najlepiej nie uruchamiać tego skryptu w ramach procesu narzędzia PowerShell, na którym Ci zależy, ponieważ nie będziesz w stanie przywrócić pierwotnego poziomu integralności.

*Listing 4.32. Sprawdzanie personifikacji tokena*

```
PS> $token = Get-NtToken -Duplicate
PS> Test-NtTokenImpersonation $token
True

PS> Set-NtTokenIntegrityLevel -IntegrityLevel Low
PS> Test-NtTokenImpersonation $token
False

PS> Test-NtTokenImpersonation $token -ImpersonationLevel Identification
True
```

Sprawdzenia te są dość proste. Najpierw uzyskano duplikat tokena bieżącego procesu i przekazano go poleceniu `Test-NtTokenImpersonation`. Wynikiem jest wartość `True` wskazująca, że możliwe byłoby podszycie się pod token na poziomie `Impersonation`. W ramach kolejnego sprawdzenia obniżono poziom integralności tokena podstawowego bieżącego procesu do poziomu `Low` i ponownie uruchomiono test. Tym razem polecenie zwraca wartość `False`, ponieważ nie jest już możliwe podszycie się pod token na poziomie `Impersonation`. Na koniec sprawdzono, czy można się podszyć pod token na poziomie `Identification`, co również powoduje zwrócenie wartości `True`.

# Praktyczne przykłady

Przeanalizujmy kilka praktycznych przykładów, aby stwierdzić, jak korzystać z różnych poleceń przedstawionych w tym rozdziale, a przydatnych w przypadku sprawdzania zabezpieczeń lub analizowania systemów.

## Znajdowanie procesów z dostępem do interfejsu użytkownika

Czasami przydatne jest wyliczenie wszystkich procesów, do których możesz uzyskać dostęp, a także sprawdzenie właściwości ich tokenów podstawowych. Może to pomóc w znalezieniu procesów działających z użyciem konkretnych kont użytkowników lub mających określone właściwości. Możesz na przykład zidentyfikować procesy z ustawioną flagą dostępu do interfejsu użytkownika. Wcześniej w rozdziale omówiono, jak sprawdzić flagę dostępu do interfejsu użytkownika w trybie izolacji. W listingu 4.33 dokonano sprawdzenia pod kątem wszystkich procesów, do których można uzyskać dostęp.

*Listing 4.33. Znajdowanie procesów z dostępem do interfejsu użytkownika*

```
PS> $ps = Get-NtProcess -Access QueryLimitedInformation -FilterScript {
    Use-NtObject($token = Get-NtToken -Process $_ -Access Query) {
        $token.UIAccess
    }
}
PS> $ps
Handle Name       NtTypeName Inherit ProtectFromClose
------ ----       ---------- ------- ----------------
3120   ctfmon.exe Process    False   False
3740   TabTip.exe Process    False   False

PS> $ps.Close()
```

Zaczęto od wywołania polecenia `Get-NtProcess`, aby otworzyć wszystkie procesy z prawem dostępu `QueryLimitedInformation`. Określono również skrypt filtrujący. Jeśli zwróci on wartość `True`, polecenie zwróci proces. W przeciwnym razie skrypt zamknie uchwyt procesu.

W skrypcie otwierany jest token procesu w celu uzyskania prawa dostępu `Query` i zwracana jest właściwość `UIAccess`. W wyniku odfiltrowano listę procesów w celu zaprezentowania tylko tych działających z tokenami dostępu do interfejsu użytkownika. Wyświetlono znalezione procesy.

## Znajdowanie uchwytów tokenów poddawanych personifikacji

Istnieje kilka oficjalnych sposobów uzyskiwania dostępu do tokena poddawanego personifikacji, takich jak użycie zdalnego wywołania procedury lub otwarcie tokena podstawowego procesu. Innym podejściem jest znajdowanie istniejących uchwytów obiektów `Token`, które można zduplikować i zastosować na potrzeby personifikacji.

Technika ta może być przydatna, jeśli korzystasz z konta użytkownika innego niż administrator, dysponującego uprawnieniem `SeImpersonatePrivilege` (jak w przypadku konta usługi, takiego jak *LOCAL SERVICE*), lub zamierzasz ocenić stopień bezpieczeństwa „piaskownicy” w celu upewnienia się, że nie może ona otworzyć bardziej uprzywilejowanego tokena i podszywać się pod niego. Z techniki tej możesz też skorzystać, aby uzyskać dostęp do zasobów innego użytkownika, czekając, aż połączy się z komputerem z systemem Windows (na przykład za pośrednictwem sieci). Jeżeli uzyskasz token użytkownika, możesz ponownie użyć jego tożsamości bez potrzeby znajomości jego hasła do konta. Listing 4.34 prezentuje prostą implementację tej koncepcji.

*Listing 4.34. Znajdowanie uchwytów tokenów z większymi uprawnieniami poddawanych personifikacji*

```
PS> function Get-ImpersonationTokens {
    $hs = Get-NtHandle -ObjectType Token  ❶
    foreach($h in $hs) {
        try {
            Use-NtObject($token = Copy-NtObject -Handle $h) {  ❷
                if (Test-NtTokenImpersonation -Token $token) {  ❸
                    Copy-NtObject -Object $token
                }
            }
          } catch {
          }
      }
  }
  PS> $tokens = Get-ImpersonationTokens  ❹
  PS> $tokens | Where-Object Elevated  ❺
```

W ramach funkcji `Get-ImpersonationTokens` za pomocą polecenia `Get-NtHandle` uzyskano listę wszystkich uchwytów obiektu typu `Token` ❶. Dalej przy użyciu polecenia `Copy-NtObject` dla każdego uchwytu podjęto próbę zduplikowania go do bieżącego procesu ❷. Jeżeli operacja powiedzie się, sprawdzane jest, czy token można skutecznie poddać personifikacji. Jeśli tak, tworzona jest kolejna kopia tokena, aby nie został zamknięty ❸.

Uruchomienie funkcji `Get-ImpersonationTokens` powoduje zwrócenie wszystkich dostępnych uchwytów tokenów, które można poddać personifikacji ❹. Dysponując tymi obiektami `Token`, można uzyskać informacje o interesujących nas właściwościach. Można na przykład sprawdzić, czy token ma zwiększone uprawnienia ❺, co może wskazywać na możliwość zastosowania tokena do uzyskania w ramach jego personifikacji dodatkowych uprzywilejowanych grup.

## Usuwanie uprawnień administratora

Uruchamiając program jako administrator, możesz zdecydować się na tymczasowe usunięcie swoich uprawnień, aby wykonać pewną operację (np. przypadkowe usunięcie plików systemowych) bez ryzyka uszkodzenia systemu. Aby zrealizować taką operację, możesz skorzystać z tej samej metody, której używa komponent UAC w celu utworzenia odfiltrowanego tokena o niższych uprawnieniach. Uruchom kod z listingu 4.35 jako administrator.

*Listing 4.35. Usuwanie uprawnień administratora*

```
PS> $token = Get-NtToken -Filtered -Flags LuaToken
PS> Set-NtTokenIntegrityLevel Medium -Token $token
PS> $token.Elevated
False

PS> "Admin" > "$env:windir\admin.txt"
PS> Invoke-NtToken $token { "User" > "$env:windir\user.txt" }
out-file : Access to the path 'C:\WINDOWS\user.txt' is denied.

PS> $token.Close()
```

Najpierw odfiltrowano bieżący token i określono flagę `LuaToken`. Flaga usuwa wszystkie grupy administratora oraz dodatkowe uprawnienia, których token limitowany nie może zawierać. Flaga `LuaToken` nie obniża poziomu integralności tokena, dlatego trzeba ręcznie ustawić poziom `Medium`. Sprawdzając, czy właściwość `Elevated` ma wartość `False`, można zweryfikować, że token nie jest już uważany za token administratora.

Aby przekonać się, jak to działa, można zapisać plik w lokalizacji dostępnej wyłącznie dla administratora (np. w katalogu *Windows*). Jeśli spróbuje się to zrealizować z użyciem tokena bieżącego procesu, operacja powiedzie się. Gdy jednak próbę wykonania operacji podejmie się podczas personifikacji tokena, zakończy się ona niepowodzeniem z błędem odmowy dostępu. Używając polecenia `New-Win32Process` narzędzia PowerShell, możesz także skorzystać z tokena, aby uruchomić nowy proces z tokenem o niższych uprawnieniach.

# Podsumowanie

W rozdziale omówiono dwa główne typy tokenów: tokeny podstawowe powiązane z procesem oraz tokeny personifikacji, które są skojarzone z wątkiem i umożliwiają procesowi tymczasowe podszywanie się pod innego użytkownika. Przyjrzeliśmy się istotnym właściwościom obu typów tokenów, takim jak grupy, uprawnienia i poziomy integralności, a także temu, jak te właściwości wpływają na tożsamość zabezpieczeń ujawnianą przez token. Przybliżono następnie dwa typy tokenów „piaskownicy" (tokeny ograniczone i *lowbox*), które aplikacje, takie jak przeglądarki internetowe i czytniki dokumentów, wykorzystują do ograniczenia szkód wynikających z potencjalnych luk w zabezpieczeniach związanych ze zdalnym wykonywaniem kodu.

Dalej zastanowiliśmy się, w jaki sposób tokeny są używane do reprezentowania uprawnień administratora, uwzględniając to, jak system Windows implementuje kontrolę konta użytkownika (UAC) oraz konta administratora z podzielonym tokenem w przypadku zwykłych kont użytkowników pulpitu. W ramach tego omówienia przyjrzeliśmy się temu, co dokładnie system operacyjny uznaje za token administratora lub token zwiększonych uprawnień.

Na koniec przedstawiono kroki związane z przypisywaniem tokenów do procesów i wątków. Zdefiniowano konkretne kryteria, które muszą być spełnione, aby zwykły użytkownik mógł przypisać token, a także wyjaśniono, czym różnią się sprawdzenia dotyczące tokenów podstawowych i tokenów personifikacji.

W następnym rozdziale zostaną omówione deskryptory zabezpieczeń definiujące, jaki dostęp zostanie przyznany do zasobu na podstawie tożsamości i grup obecnych w tokenie dostępu obiektu wywołującego.

# Skorowidz

## G



## H



## I



## J

## K



## L



## M



## N

## O



## P

## R



## S

## Ś



## T



## U

## W



## Z



## Ż


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

# ...a zatem twierdzisz, że rozumiesz jak działają zabezpieczenia Windows?

W czasach zaawansowanych cyberataków na maszyny pracujące pod kontrolą Windows opanowanie złożonych mechanizmów zabezpieczeń systemu operacyjnego to kluczowa umiejętność. Zrozumienie zasad działania niskopoziomowych implementacji systemu Windows jest konieczne, jeśli chcesz odkrywać nieznane luki w zabezpieczeniach lub chronić się przed już znanymi zagrożeniami.

Tę książkę docenią projektanci, specjaliści z zakresu metodyki DevOps i badacze zajmujący się bezpieczeństwem, którzy znajdą w niej niezrównane źródło wiedzy o kluczowych elementach systemu operacyjnego i jego słabych punktach. Poszczególne zagadnienia zilustrowano za pomocą starannie przygotowanych przykładów bazujących na narzędziu PowerShell. Przykłady te można testować i dostosowywać. Obejmują one zarówno podstawową analizę zabezpieczeń zasobów, jak i techniki zaawansowane, takie jak uwierzytelnianie sieciowe. Dzięki tej praktycznej książce przyswoisz wiedzę o tym, jak system Windows zabezpiecza pliki i rejestr, jak implementuje uwierzytelnianie lokalnie i za pośrednictwem sieci, a także od podstaw przeanalizujesz zagadnienia udzielania dostępu do zasobu.

**Ciekawsze zagadnienia:**

- architektura zabezpieczeń systemu Windows
- monitor SRM systemu Windows
- tokeny dostępu, deskryptory zabezpieczeń zasobu, kontrola dostępu i audyt
- uwierzytelnianie interaktywne i Security Account Manager
- protokoły uwierzytelniania sieciowego, w tym NTLM i Kerberos

**James Forshaw** jest ekspertem do spraw cyberbezpieczeństwa w zespole Project Zero firmy Google i niekwestionowanym autorytetem w dziedzinie zabezpieczeń systemów Windows. W ciągu ponad 20 lat wykrył setki publicznie ujawnionych luk w zabezpieczeniach systemów operacyjnych. Często prezentuje wyniki swoich badań na blogach i międzynarodowych konferencjach.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
*Sięgnij po więcej!* ▶

ISBN 978-83-289-2065-1

Cena: 149,00 zł

no starch press