

Wykorzystanie sprzętowych urządzeń szyfrujących w połączeniu z bazą Oracle

Paweł Goleń

Cryptotech

Bazy danych są częścią wielu systemów informatycznych, w których przechowywane i przetwarzane są dane. Z uwagi na fakt, iż dane te są coraz bardziej istotne, konieczne staje się wykorzystanie bardziej zaawansowanych technik ich ochrony. Jednym ze sposobów na zwiększenie bezpieczeństwa danych jest wykorzystanie modułów HSM (Hardware Security Module). Prezentacja ta omawia wykorzystanie modułów HSM w następujących scenariuszach:

- ¹ szyfrowanie kolumn zawierających istotne dane w bazie,
- ¹ wykonywanie części aplikacji w module HSM,
- ¹ szyfrowanie end-to-end między klientem a bazą danych,
- ¹ akceleracja SSL.

W przypadku większości scenariuszy przedstawione zostaną zwykle stosowane rozwiązania, wykazane ich słabości i potencjalne punkty ataku, a następnie zaprezentowane rozwiązanie alternatywne wykorzystujące moduł HSM.

Informacja o autorze:

Paweł Goleń – konsultant e-security. Posiada tytuł Microsoft Certified Professional, Microsoft Certified System Administrator oraz Citrix Certified Administrator. Specjalista w zakresie Windows NT/2000/XP/2003, Linux oraz bezpieczeństwa sieci LAN i WAN. Współautor książek „Windows NT 4.0 – bezpieczny system operacyjny” i „Windows 2000 – bezpieczny system operacyjny”, „Windows XP - bezpieczny system operacyjny”. Autor wielu artykułów i prezentacji na konferencjach poświęconych bezpieczeństwu IT.

Czy dane w bazie danych są bezpieczne?

O tym, że dane składowane w systemach mają wymierną wartość wiadomo już od dawna, dlatego też mechanizmy zabezpieczeń baz danych mają dość długą historię. Praktycznie każdy producent baz danych zapewnia swoich użytkowników o rozbudowanych funkcjach zabezpieczeń zaimplementowanych w swoim produkcie. Znane jest także hasło marketingowe Oracle „nie możesz się włamać, nie możesz jej złamać”. Znana jest także rzeczywistość, wystarczy zapoznać się z działem *vulnerabilities* na SecurityFocus (<http://online.securityfocus.com>) dotyczącym Oracle, MS SQL, MySQL, Postgres lub dowolnej innej bazy danych. Nie można zapominać, że system bazodanowy jest niczym innym, jak dużym kawałkiem kodu tworzonym przez człowieka, który może popełniać błędy. Oczywiście mechanizmy zawarte bezpośrednio w bazie danych są bardzo potrzebne, pozwalają stworzyć podstawy bezpieczeństwa danych w niej przechowywanych, w niektórych sytuacjach jest to jednak niewystarczające. Baza danych nie jest celem samym w sobie. Jest ona częścią większych rozwiązań, w których dane są przechowywane i przetwarzane. Zwiększa to w znaczący sposób ilość potencjalnych zagrożeń dla bezpieczeństwa danych, wystarczy zapoznać się choćby z atakami typu „SQL injection”, dzięki którym intruz może zobaczyć dane, których widzieć nie powinien zgodnie z zamierzeniem programisty, ale które może zobaczyć w wyniku jego błędu. W dalszej części tej prezentacji omówionych zostanie kilka przykładowych rozwiązań, które w znaczący sposób zwiększają bezpieczeństwo danych w trakcie ich przetwarzania i przechowywania.

Bezpieczeństwo danych – jak je zwiększyć?

Baza danych zawiera dużą ilość różnych informacji. Część z tych informacji nie jest krytyczna, jednakże w niektórych kolumnach znajdują się informacje o naprawdę bardzo dużym znaczeniu. Co można zrobić w celu ochrony takich danych, co oferują systemy bazodanowe w swych standardowych konfiguracjach? Większość systemów pozwala na wykorzystanie następujących mechanizmów:

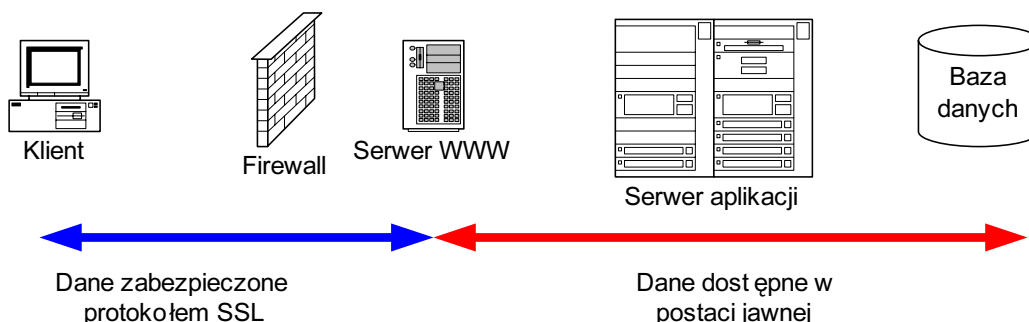
- Silne uwierzytelnianie użytkownika
- Precyzyjny system kontroli dostępu
- Mechanizmy śledzenia danych
- Szyfrowanie przesyłanych danych

Co dają poszczególne elementy mechanizmów zabezpieczeń? Dzięki silnemu uwierzytelnieniu użytkownika system (bazodanowy, lub bardziej kompletne rozwiązanie) może zweryfikować jego tożsamość. Znając tożsamość użytkownika możliwe jest wymuszenie przestrzegania praw dostępu do poszczególnych danych składowanych w bazie danych, a mechanizmy audytu pozwalają śledzić wykorzystanie przyznaných uprawnień. Połączenie tych wszystkich elementów pozwala osiągnąć stosunkowo wysoki poziom bezpieczeństwa danych przechowywanych w bazie danych, jednakże nie rozwiązuje wszystkich problemów. Najprostszym przykładem zagrożenia dla poufności są konta DBA, które uzyskują dostęp do wszystkich danych. Do danych z poufnych kolumn lub należących do innych użytkowników można uzyskać również dostęp przez różnego typu ataki, choćby ataki „SQL injection”. Dlaczego nie można rozwiązać tych problemów przy pomocy mechanizmów kontroli dostępu? Między innymi z tego powodu, że często w trakcie interakcji złożonego systemu z bazą danych wykorzystywana jest tożsamość jednego użytkownika. Dlaczego? Raczej trudno wyobrazić sobie, by dla każdego z klientów banku tworzone było osobne konto użytkownika w bazie danych, oddzielne tabele lub widoki, oddzielnie ustawiane uprawnienia. Użytkownik przedstawia się aplikacji swoimi danymi identyfikacyjnymi, następnie ta aplikacja wiąże dane przechowywane w bazie z identyfikatorem uwierzytelnionego w danej sesji użytkownika i prezentuje mu jego dane, co nie oznacza, że aplikacja ta nie ma w tej samej chwili dostępu do danych innego użytkownika. Co może się stać, jeśli użytkownik będzie mógł ingerować w sposób wykonywania się aplikacji, na przykład modyfikując „swoją” identyfikator, na podstawie którego z dużym prawdopodobieństwem wybierane są dane do niego „przynależne”. Skoro nie można

wykorzystać standardowych mechanizmów kontroli dostępu, czy istnieje inna droga zabezpieczenia danych przechowywanych w bazie przed niepowołanym dostępem? Jeżeli nie można zabezpieczyć danych przed niepowołanym odczytem, to można spowodować, by dane te były całkowicie niezrozumiałe dla nieuprawnionego użytkownika. Sensownym wyborem jest więc dołączenie mechanizmów szyfrowania przechowywanych danych. Szyfrowanie takie powinno mieć charakter selektywny, gdyż w innym przypadku może dojść do niepotrzebnego ograniczenia funkcjonalności bazy danych. Oznacza to, że szyfrowane powinny być naprawdę istotne dane. Dlaczego nie szyfrować wszystkiego? Z tej prostej przyczyny, że bazy danych wykonują wiele operacji na danych, wiążą je, przetwarzają. W przypadku, gdyby wszystkie dane w bazie były w postaci zaszyfrowanej każda taka operacja byłaby znacznie bardziej złożona obliczeniowo, a co za tym idzie wydajność całego systemu mogłaby spaść do nieakceptowanego poziomu. Należy także odróżnić szyfrowanie samej bazy danych (plików) od szyfrowania danych w bazie. Szyfrowanie samej bazy danych wykonywane może być z założeniem, że pliki mogą wpaść w niepowołane ręce (na przykład ktoś może wejść w posiadanie backupu). Z punktu widzenia samej bazy jednak takie szyfrowanie jest pomijalne, gdyż serwer widzi dane już odszyfrowane, w związku z czym takie działanie w żadnym stopniu nie zabezpiecza przed atakami w warstwie aplikacji. Baza Oracle 9i (a także niektóre wersje Oracle 8) posiada natywne mechanizmy szyfrowania danych w bazie implementowane poprzez pakiet DBMS_OBFUSCATION_TOOLKIT. Mechanizmy oferowane przez Oracle pozostawiają jednak wiele do życzenia, jeśli chodzi o zarządzanie wykorzystywanymi kluczami. To autor aplikacji musi rozwiązać problem generowania kluczy, składowania i ich odzyskiwania. Nie są to zadania trywialne, lecz ciężkie do bezpiecznej i efektywnej realizacji. Co więcej oparcie się w tym przypadku jedynie na rozwiązaniach programowych może stworzyć złudne poczucie większego bezpieczeństwa, choć w praktyce dane nadal są zagrożone. Połączenie mechanizmów wbudowanych w bazę Oracle ze sprzętowymi modułami HSM w znaczny sposób zwiększa ich przydatność poprzez:

- Zapewnienie bezpiecznych mechanizmów zarządzania kluczami
- Zapewnienie bezpieczeństwa wykorzystanych kluczy
- Przeniesienie operacji kryptograficznych do modułu HSM
- Wykonanie części kodu w module HSM

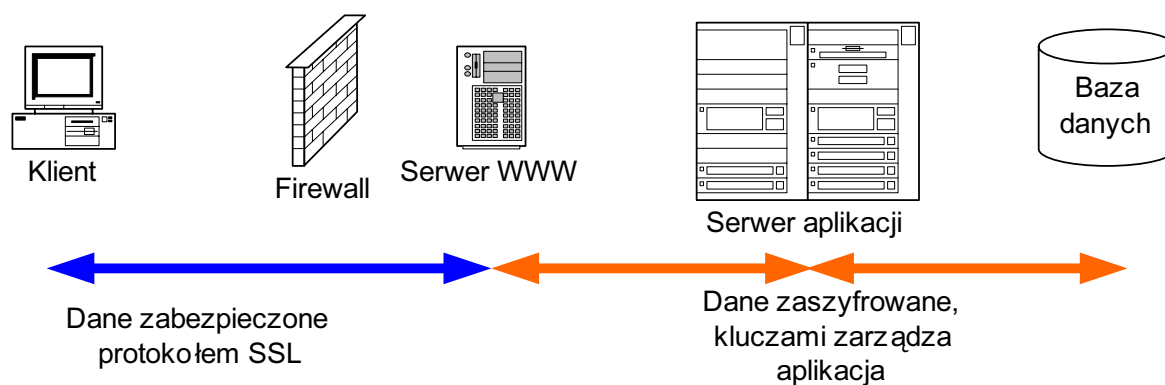
Aby lepiej zrozumieć przewagę rozwiązań sprzętowych można posłużyć się przykładem rozwiązania, w którym użytkownik poprzez bezpieczny kanał SSL przesyła do aplikacji pracującej na serwerze swoją nazwę użytkownika oraz hasło. Kanał SSL kończy się na serwerze WWW, w trakcie dalszego przetwarzania dane przesłane przez użytkownika dostępne są w formie jawnej. W dalszym etapie aplikacja ma dokonać sprawdzenia hasła podanego przez użytkownika. W przypadku, gdy hasło przechowywane jest w bazie w sposób jawny po prostu następuje porównanie zapisanej wartości z hasłem przekazanym przez użytkownika.



Rysunek 1 – sytuacja, gdy hasło przekazywane jest w postaci jawnej

Hasło jest jednak jedną z informacji, które należy uznać za poufne, w związku z czym przechowywanie go w formie jawnej nie jest dobrym pomysłem. Jak może więc w sposób bezpieczny przechowywać hasło? Można zastosować co najmniej kilka sposobów jego zabezpieczenia przed

poznaniem przez osoby niepowołane (choćby DBA). Od dawna każdy poważny system operacyjny nie przechowuje hasła w postaci jawnej lecz w postaci pewnego skrótu kryptograficznego obliczonego według określonego algorytmu. Ponieważ systemowi operacyjnemu nie jest zwykle potrzebna znajomość jawnej formy hasła algorytm ten jest zwykle funkcją jednokierunkową, a więc nie można na podstawie skrótu obliczyć jawnej postaci hasła. Gdyby hipotetyczny system składował hasło użytkownika pod postacią skrótu kryptograficznego szyfrowanie tej konkretnej porcji danych nie byłoby konieczne, jednakże trzeba zwrócić uwagę, że przykładowo w wielu systemach bankowych konieczna jest możliwość uzyskania hasła w formie jawnej. Wystarczy przypomnieć sobie o różnych serwisach transakcyjnych, do których logowanie odbywa się przez wpisanie kilku wrywkowo wybranych znaków hasła, oznacza to, że w systemie hasło musi być dostępne w formie pełnej. Wynika z tego konieczność zabezpieczenia hasła przy pomocy algorytmów odwracalnych, a więc mówiąc prościej – zaszyfrowanie tych danych przed zapisaniem ich do bazy danych. Dzięki szyfrowaniu osoba, która nawet będzie w stanie przeczytać zawartość bazy danych nie będzie w stanie odczytać hasła, o ile nie wejdzie w posiadanie stosownego klucza. I tutaj zaczyna się problem – jak zabezpieczyć klucz?



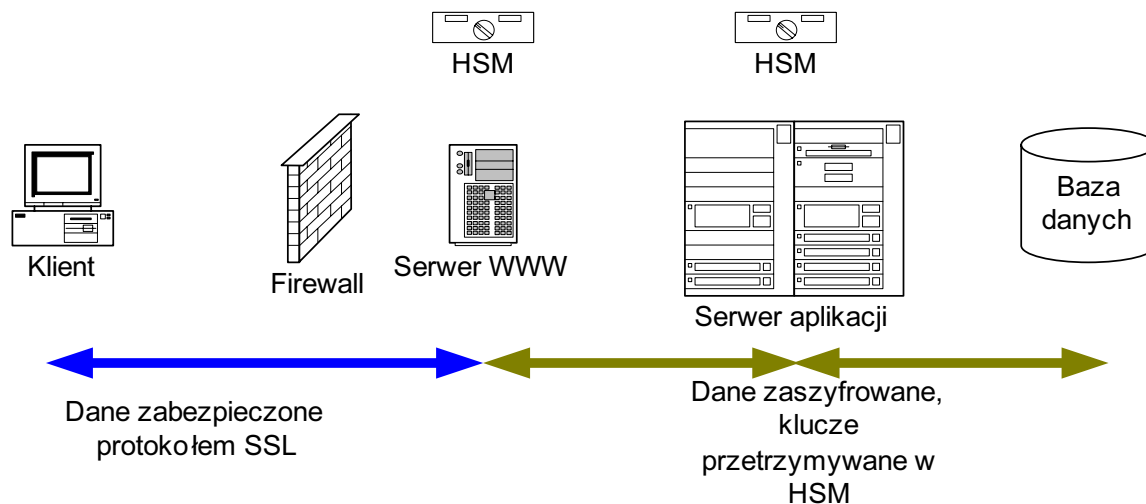
Rysunek 2 - kluczami szyfrowania zarządza aplikacja

W przypadku, gdy klucz taki zapisany jest w oprogramowaniu to poziom oferowanego bezpieczeństwa jest co najmniej dyskusyjny ponieważ istnieje możliwość uzyskania klucza wykorzystanego do szyfrowania danych w bazie danych i wykorzystanie go do odszyfrowania innych, być może jeszcze bardziej interesujących informacji przechowywanych w bazie danych. Oczywiście zakłada to, że intruz może działać na maszynie, na której uruchomiona jest ta aplikacja, co często wydaje się mało prawdopodobne w przypadku, gdy główna logika aplikacji znajduje się na serwerach back-endowych. Mało prawdopodobne, ale nie niemożliwe. Trzeba także pamiętać, że intruz nie zawsze znajduje się na zewnątrz, według różnych statystyk ilość ataków „ze środka” zamyka się w granicach od 30% do 70% wszystkich stwierdzonych naruszeń bezpieczeństwa, co jest dość pokaźnym wskaźnikiem, w związku z czym zagrożenia takiego ignorować całkowicie nie można. Dostęp do systemu często mają różni ludzie, choćby zajmujący się wsparciem technicznym sprzętu i oprogramowania. Ich także można traktować jako potencjalne źródło naruszenia bezpieczeństwa systemu. Co można zrobić, by system omówiony wcześniej był bardziej bezpieczny? Przede wszystkim należy zdać sobie sprawę, z dwóch możliwych typów ataku. Po pierwsze ktoś może przechwycić niezaszyfrowane dane po wyjściu z kanału SSL, po drugie możliwe jest uzyskanie klucza szyfrowania poprzez analizę aplikacji realizującą logikę systemu. Oba przypadki są niebezpieczne, w związku z tym narzucają się dwie rzeczy mające na celu podniesienie bezpieczeństwa:

- Szyfrowanie danych po opuszczeniu kanału SSL
- Zmiana sposobu przechowywania klucza wykorzystywanego do szyfrowania danych

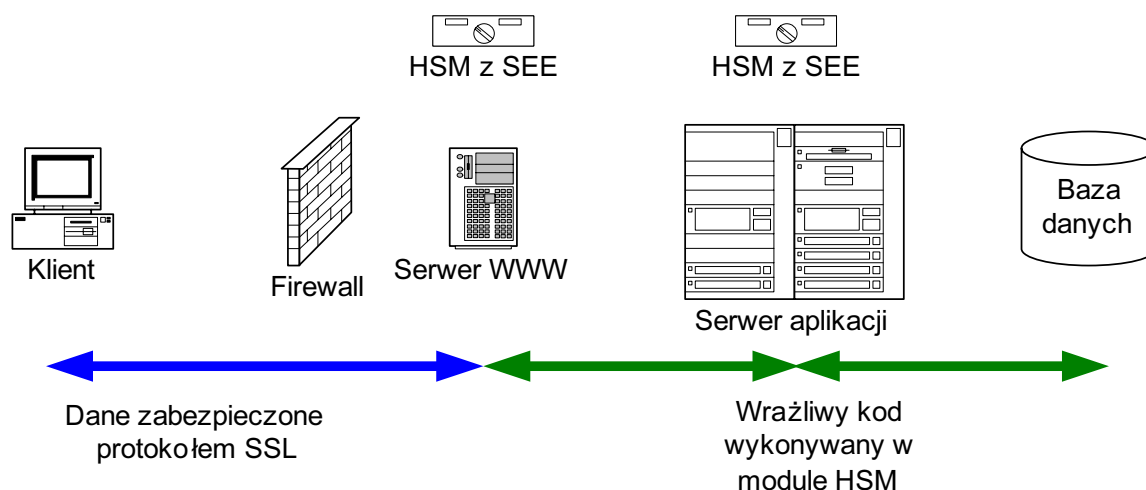
Tutaj mogą zrodzić się dwa pytania – w jaki sposób szyfrować dane i w jaki sposób przechowywać klucze wykorzystywane do szyfrowania. Dla uproszczenia rozważań założmy, że dane wychodzące z kanału SSL szyfrowane są przy użyciu szyfrów symetrycznych i przekazywane

dalej, a więc problem sprowadza się do bezpiecznego sposobu korzystania z kluczy szyfrowania. Samo lepsze zabezpieczenie klucza w trakcie jego składowania, pomijając metodę tego zabezpieczenia, nie jest tutaj rozwiązaniem wystarczającym. Klucz co prawda może być przechowywany bezpiecznie, choćby w urządzeniu HSM, jednakże przychodzi chwila, w której musi on zostać wykorzystany.



Rysunek 3 - klucze bezpiecznie przechowywane w module HSM

W chwili, kiedy aplikacja korzysta z klucza może on zostać przechwytyony przez intruza działającego na tej maszynie, z tego też powodu wskazane jest, by urządzenie HSM nigdy nie przekazywało na zewnątrz kluczy. Sytuacja taka może być jednak nieunikniona w przypadku, gdy moduł HSM dołączany jest do starszego systemu, którego nie można przekompilować w taki sposób, by w pełni wykorzystywał funkcje HSM, lub gdy migracja przebiega etapami. Nawet jeśli urządzenie HSM nie przekazuje nigdy klucza bezpośrednio do aplikacji, to oprogramowanie zawsze może zostać zmodyfikowane, ponieważ nie pracuje w środowisku, które z całkowitą pewnością wykluczałoby takie działania. Innym sposobem uzyskania dostępu do przetwarzanych przez aplikację danych to debugowanie programu lub wykonanie zrzutu wykorzystywanej przez nią pamięci. W krańcowych przypadkach można spróbować zawiesić cały system operacyjny i następnie analizować dane zapisane w plikach wymiany. Oczywiście nie jest to zadanie trywialne, ale możliwe do zrealizowania. W efekcie zastosowania modułu HSM klucz rzeczywiście może być dobrze chroniony w trakcie przechowywania i niemożliwy do pozyskania w trakcie jego wykorzystania, ale w dalszym ciągu intruz może poznać treść poufnych danych w trakcie ich przetwarzania w systemie. Optymalnym rozwiązaniem byłoby zastosowanie swoistej „czarnej skrzynki”, do której trafiają zaszyfrowane dane i w której wykonywane są wszystkie konieczne operacje na nich. W ten sposób dane te znajdowałyby się w postaci jawnej jedynie w owej „czarnej skrzynce” i nigdy nie byłyby dostępne w formie jawnej umożliwiającej ich przechwycenie. Co więcej klucze wykorzystywane do szyfrowania danych nie opuszczałyby urządzenia, co także zmniejsza w znaczący sposób ryzyko ich przechwycenia. Rolę owej „czarnej skrzynki” pełnić może moduł HSM poszerzony o możliwość uruchamiania w nim programów.



Rysunek 4 - wykonanie kodu w module HSM

W takim przypadku w trakcie tworzenia aplikacji oznaczają się fragmenty kodu jako niewralgiczne. Fragmenty te wykonywane są wewnątrz urządzenia HSM, dzięki czemu nie istnieje możliwość ingerencji w środowisko wykonania kodu. Ponieważ kod taki jest podpisywany nie istnieje możliwość modyfikacji samego kodu. Taka akcja powoduje bowiem nieważność podpisu potwierdzającego jego integralność. Kod z nieprawidłowym podpisem nie zostanie uruchomiony w module HSM, nie będzie mógł uzyskać dostępu do kluczy, a w związku z czym nie będzie w stanie odszyfrować danych. Wykorzystanie takich mechanizmów nie ogranicza się jedynie do ochrony danych w bazie danych. W przykładowym systemie należy chronić dane po opuszczeniu bezpiecznego kanału SSL. Ponieważ serwer WWW jest bezpośrednio wystawiony na ataki ze strony Internetu stopień ryzyka w jego przypadku jest stosunkowo wysoki. Jeśli cały system zabezpieczeń koncentrowałby się na serwerze aplikacji i bazie danych wówczas najsłabszym punktem stałby się właśnie narażony na bezpośrednie ataki serwer WWW. Jak zwiększyć jego bezpieczeństwo? Po pierwsze wykorzystać można HSM jako akcelerator SSL, w związku z czym kanał SSL kończy się w module HSM, a nie na serwerze. Osadzając odpowiednią aplikację można już w module HSM ponownie szyfrować część istotnych danych przed przesłaniem ich danych. Dzięki temu na serwerze WWW podobnie jak w przypadku serwera aplikacji, poufne dane nie będą dostępne nawet przez chwilę w postaci jawnej.

Czym szyfrowanie nie jest?

O czym należy pamiętać w takim rozwiązaniu? Przede wszystkim należy uświadomić sobie fakt, że szyfrowanie nie jest tożsame z kontrolą dostępu. Zszyfrowane dane wciąż mogą zostać odczytane przez osoby niepowołane, choć bez posiadania klucza nie będą one w stanie ich poprawnie zinterpretować, a przynajmniej nie będą w stanie tego zrobić w sensownym czasie. Dane te mogą jednak wciąż zostać usunięte (złośliwie) lub zmodyfikowane (choć nie koniecznie na dane sensowne). Z tego też powodu nie można zaniedbywać prawidłowego wykorzystania podstawowych metod zabezpieczeń bazy danych, a więc uwierzytelniania użytkownika, kontroli dostępu oraz audytu jego działań. Warto także tak projektować system, by w pełni wykorzystywał dostępne możliwości poprzez wprowadzenie większej ilości kont użytkowników do interakcji między serwerem aplikacji a bazą danych i rozdzielenie ich ról, restrykcyjne ustawienie kontroli dostępu i śledzenie zdarzeń. Do tej podstawy dołączyć można dodatkowe zabezpieczenia, czyli przykładowo szyfrowanie danych. Bez poświęcenia należytej uwagi każdemu z tych elementów stworzony system będzie systemem niepełnym i prawdopodobnie będzie oferował dodatkowe „funkcje”, których jego producenci nie zamierzali w nim umieszczać.

Czy to wszystko jest potrzebne?

Koszty włożone w podnoszenie poziomu bezpieczeństwa systemu nie przekładają się w sposób liniowy na osiągnięte rezultaty. Przedstawione tutaj scenariusze dokładnie odzwierciedlają taką zależność. Przy stosunkowo niskich kosztach można stworzyć system o znacznie lepszych parametrach bezpieczeństwa niż system wyjściowy. W przypadku, gdy uwarunkowania wymagają wyższego poziomu bezpieczeństwa można zastanowić się nad wykorzystaniem modułów HSM. Jeśli przeznaczenie systemu wymaga jeszcze większego poziomu bezpieczeństwa kolejnym krokiem jest HSM z bezpiecznym środowiskiem wykonania aplikacji. Kto narzuca wymagania odnośnie poziomu bezpieczeństwa systemu? W zależności od przeznaczenia może to być stosowna ustawa lub po prostu wymagania klienta wynikające z oceny ryzyka i ewentualnych kosztów, jakie musiałby on ponieść w przypadku utraty lub ujawnienia poufnych danych.