Aplikacje WWW - hackowanie

Jan Wróblewski

Wersja z 16 czerwca 2015

1. Wstęp
2. Injection
3. Cross-site scripting
4. Cross-site request forgery
5. Ścieżka pliku
6. Hasła
7. Spam
8. Denial of Service
9. Exploity

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Prawo

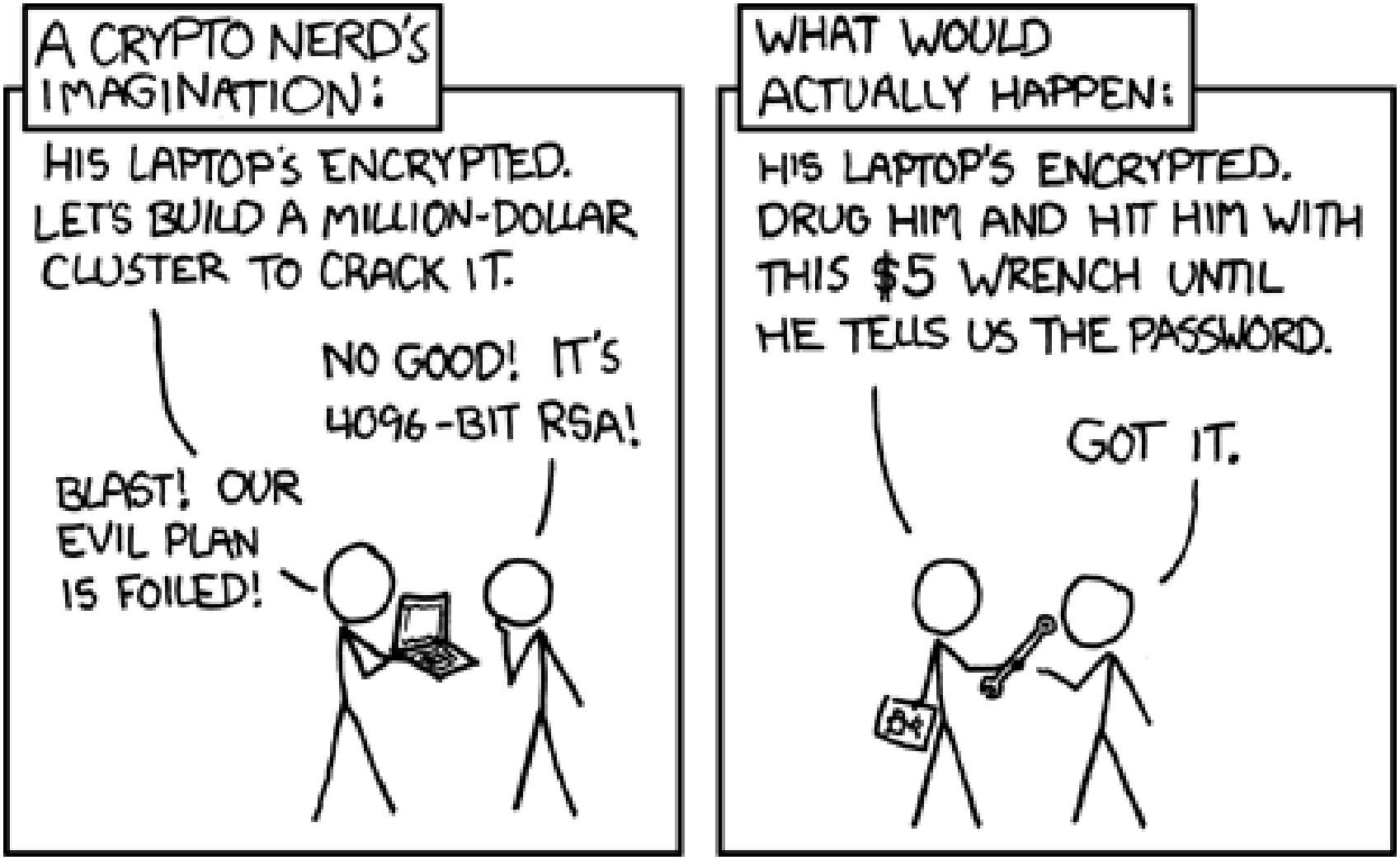
Disclaimer:

- Kodeks karny: `http://isap.sejm.gov.pl/DetailsServlet?id=WDU19970880553`, rozdział XXXIII;
- Art. 267 kodeksu karnego – włamywanie się, by zdobyć informacje jest złe;
- Art. 268, 268a, 269, 269a kodeksu karnego – niszczenie informacji lub udaremnianie dostępu do nich (DoS etc.) jest złe;
- Art. 269b – tworzenie exploitów jest złe;
- Nie jestem prawnikiem.

Generyczne porady

- Jesteśmy na przegranej pozycji – my musimy zabezpieczyć się przed wszystkim, a atakujący znaleźć tylko jedną lukę;
- Warto dostosować ilość czasu i pieniędzy poświęcaną na zabezpieczanie do wymagań aplikacji;
- Niektóre rodzaje zabezpieczeń utrudniają pracę użytkownikom (wymagania skompikowanych haseł, CAPTCHA, etc.) – warto z nimi nie przesadzać;
- Używanie sprawdzonych bibliotek i gotowych aplikacji minimalizuje ilość luk bezpieczeństwa, w szczególności związanych z obcym nam typem ataku;
- Z drugiej strony, jeżeli wiemy co robimy, to nasz własny kod closed-source może być bezpieczniejszy o tyle, że atakujący nie może zaplanować ataku "offline" i hackuje po omacku;
- Zwykle najsłabszym punktem jest czynnik ludzki.

Generyczne porady



Rysunek : http://xkcd.com/538/

Materiały

Ogólne materiały:

- `https://www.owasp.org/index.php/Top_10` – lista popularnych zagrożeń dla aplikacji WWW;
- `https://docs.djangoproject.com/en/stable/topics/security/` – zabezpieczenia wbudowane w Django.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Injection

Injection to atak, w którym dane wpisane przez użytkownika są traktowane jako część wykonywanego polecenia i przy odpowiednim ich doborze mogą zmienić semantykę programu.

Najpopularniejszą wersją jest atak SQL Injection, który polega na dodawaniu specjalnych symboli SQL do zapytania tak, by zmienić zakres danych, na jakim operujemy.

Atak SQL Injection

Załóżmy, że niezabezpieczona aplikacja wykonuje zapytanie

```
select id, topic from forum_threads where topic like
'%$query%'
```

gdzie $query to tekst wpisany w formularzu przez użytkownika.

Wywoływane zapytanie w normalnym przypadku

```
select id, topic from forum_threads where topic like
'%przepis na bigos%'
```

Wywoływane zapytanie na spreparowanym zapytaniu SQLite

```
select id, topic from forum_threads where topic like
'%' union select 1, (username || password) from users --%'
```

Atak SQL Injection

Generalnie atakując staramy się szybko zakończyć zapytanie (np. kończąc cudzysłów), dopisać naszą część zapytania (przydaje się tu `union`), a całość zakończyć SQLowym komentarzem obowiązującym do końca linii, `--`. Podobnie atakujemy inne zapytania, np. `insert`, ale generalnie jesteśmy ograniczeni typem zapytania (`insert` etc.) i składnią SQL.

Jeżeli serwis wypisuje informacje do debugowania, to jest prosto. W przeciwnym razie musimy się domyślać zapytań, ilości kolumn, typów zmiennych kolejnych kolumn etc.

Czasem połączenie SQL pozwala na wykonanie wielu zapytań oddzielonych średnikami za jednym wywołaniem funkcji. Wtedy atakujący musi jedynie być w stanie zakończyć średnikiem poprzednie zapytanie, a dalej może robić cokolwiek.

Obrona przez SQL Injection

Niskopoziomowo obrona przed SQL Injection polega na odpowiednim dodawaniu znaków escape \ przed znaki o specjalnym znaczeniu wpisane przez usera, np. przed apostrofy.

Problem w tym, że zachowanie dobrej funkcji escape'ującej może zależeć od konkretnego serwera SQL, jego wersji, a nawet kodowania znaków. Pamiętajmy również, że użytkownik może przesłać na przykład nieznormalizowany UTF-8. Dlatego to sterownik/biblioteka do konkretnej bazy danych najlepiej wie, które znaki należy escape'ować.

Obrona przez SQL Injection

W praktyce zamiast escape'ować, używamy **bindowania zmiennych** wszędzie, gdzie otrzymujemy dane od użytkownika. To zapewnia pełną separację wykonanego kodu SQL od danych użytkownika. Przy okazji tworzymy w ten sposób zapytania łatwiejsze do zoptymalizowania przez bazę danych.

Jako że Django jest oparte na technologii ORM, to naturalnie wszystkie operacje są wewnątrz bindowane. Trzeba tylko uważać, gdy się ręcznie wykonuje SQLa.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Atak XSS

Cross-site scripting to ataki wykorzystujące brak odpowiedniej transformacji danych pochodzących od użytkownika przed wyświetleniem jej na swojej stronie. Używając ataków XSS można w dowolny sposób kontrolować zawartość strony widzianą przez innych użytkowników, a nawet wywoływać u nich skrypty czy przekierowywać na inne strony.

Najpierw ustalmy cel atakującego. Może to być wywołanie dowolnego skryptu, czyli wpisanie np. `<script>...</script>`. Może to być zepsucie wyglądu strony ostylowanym HTMLem. Może to być przekierowanie użytkowników na inną złośliwą stronę za pomocą:

```
<meta http-equiv="refresh"
    content="0; url=http://evil-site.example.com/">
```

Atak XSS

Podstawowe ataki XSS są bardzo proste:

- Jeżeli strona pozwala na wpisanie dowolnego HTMLa, to atak jest trywialny;
- Jeżeli strona pozwala na umieszczenie dowolnego tekstu tylko w jakimś atrybucie HTML to wystarczy zamknąć cudzysłów/apostrof i tag, a następnie wpisać nasz HTML;
- Można próbować wpisać linki w postaci `javascript:kodJS()` do `<a href="...">` czy `<img src="...">`, które wywołują dany kod JavaScript;
- Przeglądarki są bardzo odporne na błędy, więc przetworzą dużo dziwnych konstrukcji;
- Długa lista na `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`.

Obrona przed XSS

Przed możliwością wpisania kodu HTML na stronę jest się łatwo zabezpieczyć: wystarczy zamienić < na `<`, > na `>` i & na `&`.
W przypadku tekstu wpisywanego do atrybutów (które powinny być w apostrofach lub cudzysłowiach) należy dodatkowo zamienić " na `&quot;` i ’ na `&#x27;` (nie zaszkodzi zawsze).

Większym problemem są linki. Trzeba dokładnie filtrować skąd pochodzą. Ze względu na interpretacje przeglądarek, lepiej ograniczyć dozwolone stringi niż starać się szukać wszystkich niedozwolonych oraz odpowiednio kodować (urlencode) dane występujące w linkach.

Obrona przed XSS

`https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet`

Django domyślnie zawiera podstawowe zabezpieczenie przed XSS w postaci zamiany pięciu wyżej wymienionych znaków HTML. Nie posiada jednak wbudowanego mechanizmu do ochrony linków. Dość dobre jest sprawdzenie, czy link zaczyna się od `http://` bądź `https://`.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Atak CSRF

Atak CSRF polega na wymuszeniu wykonania jakiejś akcji na użytkowniku zalogowanemu do jakiegoś serwisu.

Standardowo po zalogowaniu użytkownik posiada ciasteczko z identyfikatorem sesji bądź inną zawartością identyfikującą go na serwerze. Ciasteczka są wysyłane do serwera przy każdym połączeniu z nim, nawet tym nie zamierzonym przez użytkownika.

Gdy zalogowany użytkownik strony A wchodzi na inną stronę B, zupełnie oddzieloną od A, wciąż jest możliwe by strona B kazała jego przeglądarce wysłać dane do A. Wykorzystuje się to do ataków CSRF.

Atak CSRF

Przykład 1

Strona B może zawierać obrazek z URL z strony A. Ten URL zostanie otwarty przez przeglądarkę użytkownika niezależnie czy prowadzi do obrazka czy nie:

```
<img src="http://a.com/admin/usun-usera/?user_id=123">
```

Zauważmy, że tą metodą można od razu wysyłać formularze metodą GET - wystarczy dodać odpowiednie wartości w URL w formie `URL?key1=value1&key2=value2&...`. Dlatego formularze typu GET są z reguły mniej bezpieczne i nie powinny służyć do wprowadzania danych, tylko raczej do ich wydobywania (np. wyszukiwarka).
Poza tym ich zawartość jest widoczna w pasku adresu, czego użytkownik może nie chcieć.

Atak CSRF

Przykład 2

Strona B może zawierać skrypt JavaScript, który wysyła sfabrykowany formularz do strony A. Może to robić w niewidocznym `<iframe>` albo AJAXem i użytkownik nawet nie będzie o tym wiedział. Wystarczy formularz, który ma `action` taki, jak prawdziwego formularza na stronie A (tylko z pełnym URLem).

Przez formularz wprowadza się większość danych do stron, zatem każdy niezabezpieczony formularz na stronie A jest narażony.

Obrona przed CSRF

Trudno zabezpieczyć linki wykonujące akcję po samym otworzeniu się. Można próbować sprawdzić wartość `HTTP_REFERRER`, ale nie jest to solidna metoda. Dlatego akcje podejmowane przez uzytkowników powinny być w formie formularzy – co nie oznacza, że nie mogą być ostylowane jak zwykłe linki.

Obrona przed CSRF polega na dodaniu do pól wypełnianych formularzy sekretnej wartości wygenerowanej przez serwer przy wyświetleniu formularza (CSRF token) i sprawdzanej po otrzymaniu go. W atakach CSRF atakujący nigdy nie poznaje danych wyświetlanych użytkownikowi na oryginalnej stronie, zatem nie ma możliwości wysłania poprawnego CSRF tokena. Musiałby do tego mieć dostęp do zawartości okna z inną stroną lub do ciasteczek z innej strony, ale to zabezpieczone na poziomie samej przeglądarki.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Atak wykorzystujący ścieżkę plików

Niektóre portale udostępniają pliki generowane z danych na dysku bądź po prostu będące ręcznym odczytaniem danych z dysku i wysłaniem ich użytkownikowi. Przykładowo może być strona, gdzie wszystkie pliki uploadowane przez użytkownika trafiają do katalogu `projekt/uploads/`, zawartość katalogu jest listowana użytkownikowi i może on wyświetlić dowolny plik.

Naiwna implementacja mogłaby być taka, że użytkownik wchodząc na stronę `images/obrazek.png` otrzymuje zawartość pliku `projekt/uploads/obrazek.png`, gdzie `obrazek.png` może być dowolnym stringiem. Taką implementację można wykorzystać podając sfabrykowany URL prowadzący w efekcie do dowolnego pliku w projekcie, np. `projekt/uploads/../secret.db`.

Atak wykorzystujący ścieżkę plików

Tego typu implementacje z reguły wykorzystujemy podając ścieżkę odpowiednią ścieżkę względną. W zależności od implementacji może być trzeba podać takie znaki jak / w formie zakodowaniej (urlencode).

Oprócz takich zastosowań jak pobieranie danych z plików z katalogu, podobnie działają projekty wykonujące kod pliku o podanej nazwie. To jest jednak z mniej bezpiecznych implementacji wstawiania danych na stronę w takich językach jak PHP, która ma na celu ominięcie używania templateów.

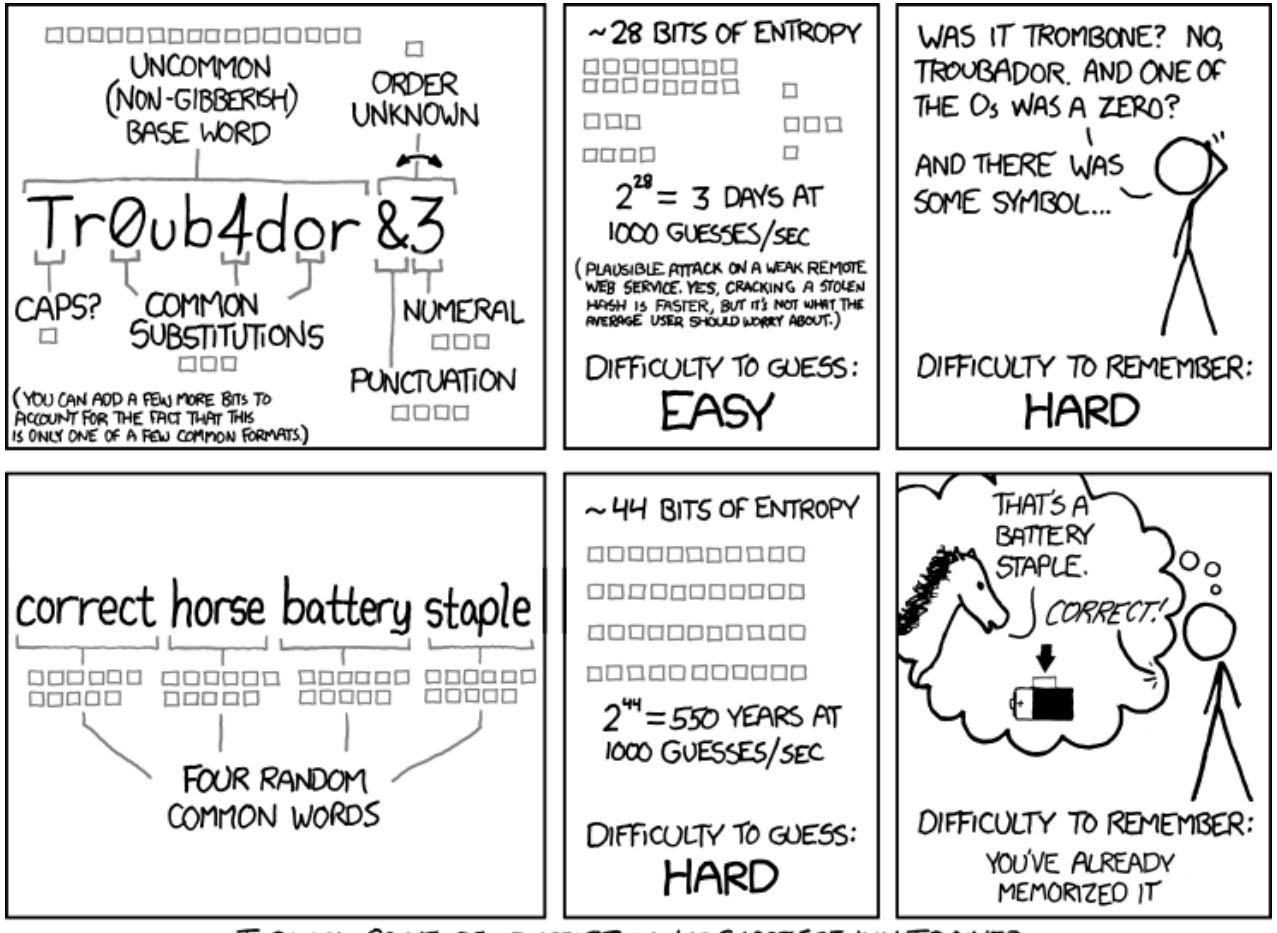
Obrona przed wykorzystaniem ścieżek plików

Obrona polega na sprawdzeniu co się zwraca, tj:

- Sprawdzeniu czy wskazywany plik rzeczywiście istnieje;
- Sprawdzeniu czy adres zawiera `..`, `/` lub inne sekwencje znaków o specjalnym znaczeniu;
- A tak najlepiej to po prostu ograniczeniu dozwolonych wejść do jakiegoś wyrażenia regularnego – metoda whitelistowania jest dużo skuteczniejsza niż blacklistowanie, bo nie wymaga znajomości wszystkich detali do bezpieczego zaimplementowania.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Faktyczna siła hasła



Rysunek : `https://xkcd.com/936/`

Przechowywanie

Warto kazać użytkownikom wprowadzać hasła pewnej sensownej długości (np. conajmniej 8 znaków). Ale nie na tym się kończy bezpieczeństwo haseł.

Haseł nie należy trzymać w postaci plaintext. Jeżeli będziemy trzymać ich hashe (np. sha1), to nawet gdy ktoś włamie się do naszej bazy danych, nie będzie znał haseł naszych użytkowników. Dzięki temu nie będzie (zbyt łatwo) się w stanie zalogować jako jakiś użytkownik, jak już naprawimy lukę w bezpieczeństwie. Chroni to użytkowników, którzy nie zmienili haseł od razu po ataku.

Przechowywanie

Jeżeli `H` jest naszą funkcją hashującą, to w praktyce możemy trochę zwiększyć bezpieczeństwo w następujący sposób:

- Tworzymy losowy string `secret` dla całej aplikacji i zapisujemy go w bezpieczym miejscu;
- Zapisujemy w bazie danych losowy string `seed(user)` dla każdego użytkownika;
- Zamiast hasła użytkownika `pass` trzymamy w bazie danych `H(secret + seed(user) + pass)`.

Dzięki temu aby złamać hasło atakiem offline to potrzeba znać `secret` i `seed(user)`, a i tak trzeba przeprowadzić ten atak osobno dla każdego użytkownika (bo są różne `seed(user)`).

Dzięki temu utrudniamy atak offline, bo nie zadziałają stablicowane wartości hasha ani nie będzie można wykonywać ataków na wszystkich użytkowników naraz.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Rodzaje ataków

Użytkownicy mogą zechcieć zaspamować nasz portal z wielu powodów:

- Chcą rozreklamować jakąś inną stronę;
- Chcą zrobić nasz system komentarzy bezużytecznym poprzez utrudnienie znalezienia sensownych komentarzy;
- Nudzi im się albo coś im się w nas nie podoba.

Sami spamerzy są też różnych rodzajów:

- Boty chodzące po różnych stronach i spamujące gdzie się da;
- Boty ustawione przez człowieka na działanie konkretnie na naszej stronie;
- Ludzie.

Rodzaje ataków

Atak polega po prostu na napisaniu skryptu wpisującego spam na stronie albo po prostu na ręcznym spamowaniu. Przykładową biblioteką używaną do automatycznej interakcji z stronami WWW jest libcurl. Znanym nam już rozwiązaniem (ale wolnym) byłaby również biblioteka selenium.

Atak będzie miał inną charakterystykę w każdym przypadku i walczy się z nim na różne sposoby.

Obrona

Z różnymi rodzajami spamerów walczymy na różne sposoby. Oto lista kilku technik:

- Popularną i skuteczną techniką walki z spamem jest ręczne zatwierdzanie komentarzy lub postów przez moderatora. Jeżeli coś jest spamem, nawet napisanym przez człowieka bez żadnych linków, to zostanie łatwo wykryte. Aby przyspieszyć pojawianie się komentarzy można automatycznie zacząć akceptować komentarze od użytkowników z już zaakceptowanymi. Minus to brak dyskusji na żywo, bo większość wypowiedzi musi być ręcznie zatwierdzana i pojawia się z opóźnieniem. Jest to też mało odporne na zmasowany atak botów, bo moderatorzy mają skończoną ilość czasu;

Obrona

- Jedną z technik chroniących przed botami jest CAPTCHA. Wymaga się od użytkownika najczęściej wpisania kodu z niewyraźnego obrazka. Ta technika ma ograniczoną skuteczność gdy nie jest dobrze zaimplementowana, bo boty mogą umieć OCR. Popularną biblioteką do tego jest reCAPTCHA `https://www.google.com/recaptcha`;
- Trikiem działającym na boty jest stworzenie dodatkowego pola formularza ostylowanego tak, by było niewidoczne dla użytkownika. Jeżeli coś jest tam wpisane to wiadomo, że jest to spamem;

Obrona

- Pozostaje technologia pozwalająca na automatyczne odfiltrowanie masowego spamu od botów od prawdziwych komentarzy. W połączeniu z moderacją znacznie zmniejsza to obciążenie moderatorów. Taka kombinacja jest stosowana np. do komentarzy w Wordpressie. Automatyczne odfiltrowywanie spamu jest zadaniem skomplikowanym. Filtrowanie po źródle i banowanie adresów IP ma ograniczoną skuteczność, ale zawsze trochę. To czego spamer nie może ukryć to faktyczną treść swojej reklamy – zatem musimy je automatycznie wykrywać.

Akismet i Sblam!

Automatycznie wykrywanie spamerskich reklam jest trudne, bo wymaga przetwarzania języka naturalnego. Do tego nadają się narzędzia używające sztucznej inteligencji lub statystyki.

To wymaga jednak sporej mocy obliczeniowej i bardzo dużej ilości danych do treningu, zatem nie jest proste do założenia na małej stronie. Między innymi dlatego powstały oferty SaaS pozwalające na sprawdzenie, czy coś jest spamem poprzez zewnętrzny serwis.

Przykładami takich serwisów są:

- Akismet – `http://akismet.com/` – popularny i wiodący w Wordpressie, ale closed source i płatny dla biznesu;
- Sblam! – `http://sblam.com/` – open-source alternatywa dla Akismeta, gdzie możemy korzystac z serwera Sblam! albo założyć własny.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Atak

Ataki DoS (Denial of Service) i wersja rozproszona z wielu komputerów naraz – DDoS (Distributed Denial of Service) – polegają na doprowadzeniu serwera do stanu, w którym nie jest dostępny dla użytkowników. Ataki DoS można podzielić na:

- Ataki polegające na saturacji łącza serwera poprzez wysyłanie na niego takiej ilości zapytań, że nie starcza łącza na obsługę prawdziwych użytkowników;
- Ataki polegające na zmuszeniu serwera do wykonywania dużej ilości kosztownych obliczeniowo zapytań zużywając całe dostępne zasoby m. in. CPU, RAM bądź ilości połączeń z bazą danych;
- Ataki exploitami polegające na wykorzystaniu luki w bezpieczeństwie do crashowania aplikacji na tym serwerze.

Te ostatnie mogą być połączone z próbą przejęcia kontroli nad serwerem.

Atak

Ataki DoS przeciążające serwer nie są skomplikowane i mogą być wykonywane nawet przez mało doświadczonych ludzi z odpowiednią aplikacją. Ponadto są bardzo efektywną metodą ataku, jeżeli zależy nam tylko na unieruchomieniu serwera. Zyskały przez to na popularności w różnego typu "akcjach" takich jak np. atak userów 4chan na tumblr, MPAA czy RIAA.

Aby wykonać taki atak DoS wystarczy dobre łącze i prosty program, który wysyła na serwer odpowiednie pakiety TCP lub UDP, nawet nie zwracając uwagi na odpowiedź.

Atak

Istotne z punktu widzena atakującego jest to, że albo wysyła się dostatecznie dużo czegokolwiek na serwer, by nie starczyło mu łącza albo by dawało się serwerowi dużo kosztownych obliczeniowo zapytań (wtedy też nie potrzeba tak silnego łącza do ataku).

DDoS to po prostu wersja ataku z wielu komputerów naraz (np. przejętych za pomocą trojana albo od innych użytkowników atakujących wspólny cel).

Bardzo silne ataki DDoS można również kupić w internecie za kwoty rzędu 50-150 złotych za miesiąc.

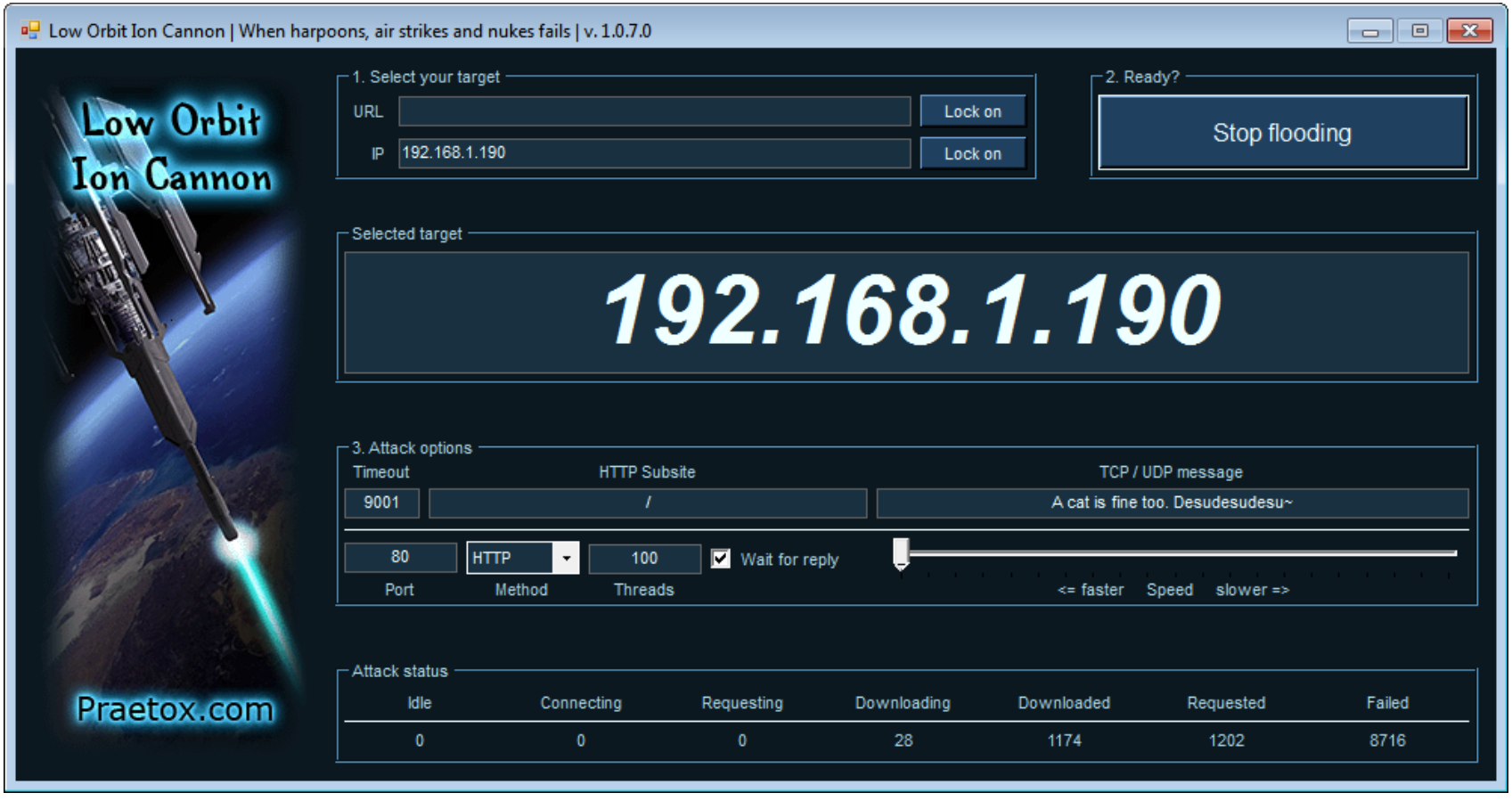
Ataki DoS przez exploit będą opisane w kolejnej sekcji.

Atak

Popularnym prostym programem do DoS jest LOIC (Low Orbit Ion Cannon), skutecznie używany przez zgromadzenia mało doświadczonych użytkowników. LOIC był użyty między innymi w ataku użytkowników 4chan na RIAA.

LOIC to prosty program do DoS napisany w C#. Niedoświadczony użytkownik może łatwo za jego pomocą zrobić generyczne ataki używające TCP, UDP albo konkretnie protokołu HTTP (przez TCP).

Atak



Rysunek : Low Orbit Ion Cannon

Atak

Formalnie LOIC służy do testowania bezpieczeństwa aplikacji i jest dostępny jako open-source na `http://sourceforge.net/projects/loic/`. Kluczowa część aplikacji z implementacją samego ataku znajduje się w krótkim pliku `XXPFlooder.cs` i polega po prostu na wysyłaniu z odpowiednią częstotliwością ustalonych pakietów na serwer.

LOIC jest dostępny przede wszystkim na Windowsa, ale można go uruchomić na Linuxie za pomocą `mono LOIC.exe`.

Uwaga: nie do wszystkich wersji LOIC jest dostępny kod źródłowy. Obecnie najnowszą wersją z udostępnionym kodem źródłowym jest 1.07.

Obrona

Na początek bardzo dobre wpisy z stackoverflow:

- `http://serverfault.com/questions/211135/how-to-prevent-a-loic-ddos-attack` (o DoS i DDoS);
- `http://serverfault.com/questions/531941/i-am-under-ddos-what-can-i-do` (o DoS zużywającym zasoby).

Obrona

Generalnie obrona przed atakami DoS jest bardzo trudna.

- W przypadku ataków doprowadzających łącze do saturacji nic nie możemy zrobić, bo to nie my kontrolujemy łącze. Musimy poprosić naszego ISP lub firmę hostującą serwer o pomoc;
- W przypadku ataków zużywających zasoby serwera możemy ograniczyć częstotliwość przetwarzanych zapytań firewallem z tzw. traffic shaping, np. używając modułu hashlimit w iptables;
- W przypadku ataków exploitami możemy dbać o aktualizacje używanych aplikacji linuxowych, bibliotek i aplikacji WWW. Jeżeli zostaniemy zaatakowani nie spatchownym dotąd exploitem to musimy to zdebugować i jakoś zablokować to zachowanie zmieniając aplikację WWW lub dostęp do jej części.

iptables hashlimit

Przykład traffic shaping z użyciem modułu hashlimit iptables (`iptables -F` by usunąć wszystkie wpisy po eksperymentach).

Ograniczanie do 60 zapytań z danego IP

```
iptables -A INPUT -p tcp --dport 80 -m hashlimit --hashlimi
    --hashlimit-burst 300 --hashlimit-mode srcip --hashlimi
iptables -A INPUT -p tcp --dport 80 -j DROP
```

Powyższe przepuszcza do 60 pakietów na minutę (`--hashlimit-upto 60/min`) z każdego IP (`--hashlimit-mode srcip`). Pierwsze 300 (`--hashlimit-burst 300`) pakietów w serii zostanie potraktowanych specjalnie i zostanie przepuszczone.

man: `http://ipset.netfilter.org/iptables-extensions.man.html`.
Informacje: `http://tlfabian.blogspot.com/2014/06/`

Pośrednik

Możemy przed atakami DoS się dobrze zabezpieczyć dodając pośrednika pomiędzy naszym serwerem a użytkownikami.

Taki pośrednik odpowiada za przekazywanie żądań użytkowników na nasz serwer. W ten sposób dopóki adres IP serwera będzie nieznany to będzie on bezpieczny.

Pośrednik

Przykładami takich pośredników są:

- Cloudflare – `https://www.cloudflare.com/`, mają darmowy plan, głównie chronią przed DDoS;
- Akamai – `http://www.akamai.com/`, pod Cloud Security Solutions, mają dużo innych usług.

Generalnie tego typu pośrednicy oprócz ochrony przed DDoS za pomocą swoich dobrych firewalli i łączy często również oferują:

- Przyspieszenie ładowania się strony za pomocą CDN (Content Delivery Network) – przesyłania danych z serwerów bliskich lokalizacji userów;
- Statystyki odwiedzin wraz z informacjami o użytkownikach;
- Inne usługi sieciowe, np. DNS.

1 Wstęp
2 Injection
3 Cross-site scripting
4 Cross-site request forgery
5 Ścieżka pliku
6 Hasła
7 Spam
8 Denial of Service
9 Exploity

Atak

Exploity to gotowe programy służące do wykonania kodu wykorzystującego lukę w bezpieczeństwie. Polegają one na wykorzystywaniu konkretnych luk bezpieczeństwa w (zwykle opublikowanych) kodach źródłowych aplikacji. Exploity są bardzo zróżnicowane i specyficzne dla konkretnej aplikacji, więc skupimy się na konkretnym exploicie wykorzystującym lukę bezpieczeństwa w Drupalu – aplikacji CMS (Content Management System) pozwalającej na łatwe zrobienie strony www.

Wykorzystamy lukę bezpieczeństwa w Drupalu w wersji mniejszej niż 7.16 – `https://www.drupal.org/node/1815912`. Została ona już dawno spatchowana pod koniec roku 2012, ale wciąż możemy pobrać starego Drupala do testów.

Atak

Stary skrypt instalacyjny Drupala działał tak, że jeżeli nie było połączenia z bazą danych (np. bo nie było zainstalowane) to pozwalał na instalację serwisu. Jeżeli połączenie z bazą danych było to skrypt instalacyjny wyświetla komunikat, że Drupal już jest zainstalowany.

W założeniu administrator uruchamia instalator Drupala za pierwszym uruchomieniem strony, a później strona zawsze już wykrywa, że jest skonfigurowana.

Atak

Exploit wykorzystuje fakt, że jeżeli Drupal jest pod dużym obciążeniem (np. w trakcie ataku DoS) to nie zawsze udaje się nawiązać połączenie z bazą danych, bo baza danych umie obsłużyć tylko niewielką ilość równoległych połączeń. Zatem po prostu DoSując Drupala można ponowić instalację przekonując instalator, że połączenie z bazą danych nie zostało jeszcze zainstalowane.

Ale nie na tym koniec. Można częściwo zniszczyć stronę np. zmieniając bazę na nową SQLite i kontrolując ją aż administrator się zorientuje. Ponoć można w jakiś sposób uruchomić dowolny kod PHP (pewnie zapisując coś ciekawego w settingsach), a zatem także funkcję `exec` wykonującą dowolny inny program, np. shella. A wtedy możemy wszystko.

Opis exploita: `http://heine.familiedeelstra.com/drupal-7-installer-vulnerability`.

Obrona

W tym konkretnym przypadku można się obronić kasując lub blokując dostęp do `install.php` po instalacji Drupala.

Przed exploitami zabezpieczają nas przede wszystkim producenci oprogramowania, z którego korzystamy. My możemy dbać o regularne aktualizacje i regularne sprawdzanie w logach czy nie dzieje się nic podejrzanego.

Gdy jest problem, a aplikacja jest w najnowszej wersji to musimy wykazać się inwencją i albo naprawić samemu aplikację albo jakoś zablokować dostęp do zasobu uzywanego do tego ataku (np. konkretnej podstrony).