Praktyczna strona modeli zagrożenia

Jako programista i inżynier bezpieczeństwa koncentrujący się raczej na praktycznej stronie ataków i obrony przed nimi model ataku kojarzy mi się często z różnymi abstrakcyjnymi prezentacjami w PowerPoincie, które – zamiast informować – głównie ładnie wyglądają albo próbują sprzedać jakiś produkt.

CO TO JEST MODEL ATAKU I PO CO O NIM MYŚLEĆ?

Nie ma systemu idealnie bezpiecznego. Nie można też wprowadzać zabezpieczeń, nie zastanawiając się, przed czym mają chronić użytkowników[1]. Myślenie powinno zaczynać się od drugiej strony: wyobrażamy sobie hipotetycznego atakującego i zastanawiamy się, w jaki sposób może nam zagrozić – oraz jak można mu to uniemożliwić. Inaczej mówiąc, zakładamy jakąś hipotetyczną sytuację (np. przestępca uruchomił złośliwe oprogramowanie na komputerze ofiary) i zastanawiamy się, jakie w najgorszym przypadku mogą być tego konsekwencje z naszej perspektywy.

Dzięki temu możemy dobrze przygotować się na najbardziej prawdopodobne ataki i mieć gotowy plan poradzenia sobie z nimi. I odwrotnie – nie spędzimy czasu na pisaniu środków bezpieczeństwa, które nie wniosłyby nic w naszej sytuacji. A przede wszystkim szybko dojdziemy do wniosku, że żaden system nie jest **całkowicie** bezpieczny – więc możemy się tylko bronić przed konkretnymi atakami.

PRZYKŁAD 1: LOGOWANIE SIĘ DO KONTA BANKOWEGO

Jako pierwszy przykład weźmy znaną wszystkim sytuację – logowanie się do konta bankowego. Zazwyczaj mówiąc, że strona internetowa ma luki bezpieczeństwa, mamy na myśli to, że atakujący może przeprowadzić bezpośrednio na niej szkodliwe działanie – na przykład włamać się do panelu administratora, wykonać własne zapytania do bazy danych (tzw. SQL injection) albo własny kod na serwerze (tzw. RCE – *Remote Code Execution*). Jest to pierwszy i najbardziej oczywisty sposób ataku. Byłby on katastrofalny w skutkach, więc niektóre strony są obwarowane przeróżnymi zabezpieczeniami mającymi przed tym chronić. Jednak jest też wiele mniej oczywistych zagrożeń:

Atakujący podsłuchujący ruch sieciowy (tzw. MITM – *Man In The Middle*) – to jeden z najstarszych problemów bezpieczeństwa w Internecie, na szczęście już dawno rozwiązany dzięki wprowadzeniu protokołu HTTPS. Przy korzystaniu z nieszyfrowanego protokołu HTTP osoba kontrolująca jeden z routerów po drodze (np. ISP, jeden z domowników w sieci lokalnej albo właściciel kafejki internetowej) może swobodnie podsłuchiwać i podmieniać ruch między klientem a serwerem. Pozwala to między innymi poznać hasła, przejąć ciastka logowania oraz wykonywać dowolne akcje jako właściciel konta. Zabezpieczeniem jest oczywiście szyfrowanie i weryfikacja ruchu.

Złośliwe oprogramowanie podsłuchujące ruch sieciowy (tzw. MITB – *Man In The Browser*) – niestety, wszystkie gwarancje dawane przez HTTPS zdają się na nic, w momencie kiedy na komputerze ofiary uruchomione jest złośliwe oprogramowanie. W takiej sytuacji atakujący może podsłuchiwać ruch już po odszyfrowaniu albo jeszcze przed zaszyfrowaniem. Bardzo ciężko, ze strony banku, uchronić klienta przed takim zagrożeniem. Popularne rozwiązania to sprawdzanie, czy nie ma złośliwych skryptów wstrzykniętych na stronę, oraz wykrywanie innych prób manipulacji treścią – ale generalnie w takiej sytuacji może pomóc tylko dobry program antywirusowy – albo klient dbający o czystość swojego systemu.

Phishing – jedna z najbardziej popularnych metod wyłudzania, na którą jednak ludzie ciągle się łapią. Najlepszą, ale nie rozwiązującą problemu całkowicie metodą zapobiegania jest edukacja. Mimo wszystko warto pomóc użytkownikowi od strony technicznej. Na przykład często przy logowaniu na prawdziwej stronie wyświetlany jest obrazek unikalny dla każdego klienta. Kiedy przy logowaniu nie wyświetli się odpowiedni obrazek, jest to prawie pewny znak, że ktoś próbuje zaatakować nasze konto.[2]

Teraz dla odmiany kilka kontrprzykładów: środków obrony, dla których ciężko wymyślić praktyczny atak uzasadniający je:

Maskowane hasła – w teorii maskowane hasła bronią przed keyloggerami (czyli złośliwym oprogramowaniem nagrywającym naciśnięte klawisze). Z tego powodu zostały zaadaptowane przez wiele banków, szczególnie że w różnych „rankingach bezpieczeństwa" przyznawane są za to dodatkowe punkty. Tymczasem w praktyce to właściwie czysta szopka bezpieczeństwa. Po pierwsze: skoro ktoś może nagrać jedno logowanie, może równie dobrze zrobić to kilka razy i zdobyć wszystkie potrzebne znaki. Po drugie: jak wspominaliśmy, lokalny atakujący w pełni kontroluje zawartość przeglądarki, więc może po prostu... poprosić użytkownika o podanie brakujących mu znaków (badania pokazują, że w praktyce wielu ludzi wpisuje swoje hasło, nawet jeśli wszystkie pola maskowanego hasła są wymagane). I wreszcie: atakujący nie potrzebuje wcale znać hasła. Po tym, jak klient już się zaloguje, może po prostu przejąć jego sesję.

1. Cóż… Właściwie to można. W efekcie zamiast bezpieczeństwa zwiększany jest tylko poziom irytacji klientów albo programistów.

2. Niestety jest to dalekie od pełnego rozwiązania problemu phishingu. W końcu przestępcy mogą skorzystać z loginu (podanego przez użytkownika), żeby samemu zapytać prawdziwy system o odpowiadającą klientowi grafikę. Bank z kolei może próbować bronić się przed tym, implementując odpowiednie heurystyki blokujące podejrzanych użytkowników w systemach antyfraudowych – kontynuując wyścig zbrojeń pomiędzy stronami. Tak czy inaczej podwyższa to poprzeczkę dla przestępcy, a że z „biznesowego" punktu widzenia lepiej słabo podszywać się pod pięć banków niż idealnie pod jeden, takie detale często bywają przeoczone lub ignorowane.

Zapraszamy na autorskie szkolenia
z zakresu bezpieczeństwa IT

{ Bezpieczeństwo aplikacji WWW }

{ Offensive HTML, SVG, CSS and other Browser-Evil }

{ Wprowadzenie do bezpieczeństwa IT }

{ Szkolenie przygotowujące do egzaminu CEH }
(Certified Ethical Hacker)

www.securitum.pl/oferta/szkolenia

Patroni medialni: sekurak.pl rozwal.to

Hashowanie haseł – popularną techniką zwiększającą bezpieczeństwo jest trzymanie w bazie tylko skrótów haseł (albo jeszcze lepiej – używanie odpowiedniej PBKDF). Przebiło się to nawet do powszechnej świadomości i wycieki baz z hasłami czystym tekstem są słusznie zdecydowanie piętnowane w Internecie (a od niedawna w Unii Europejskiej można za to też całkiem solidnie dostać po kieszeni). Często pojawia się pytanie, w jaki sposób banki to robią, skoro używają haseł maskowanych[3]. Okazuje się jednak, że w przypadku maskowanych haseł często hasła wcale nie są hashowane – i nie jest to wbrew pozorom problem. Warto zastanowić się, przed czym się bronimy. Hashowanie haseł minimalizuje skutki wycieku bazy danych, bo użytkownicy używający tego samego hasła na wielu stronach nie ucierpią aż tak dotkliwie (przy użyciu tzw. soli – z ang. *salt* – każdy hash będzie musiał być łamany osobno na kosztownych klastrach obliczeniowych). W tym przypadku jednak, po pierwsze, o ile używanie tego samego hasła w wielu miejscach to zły pomysł, to już używanie identycznego hasła do banku i innych stron jest szaleństwem. Po drugie, nawet najlepsza funkcja skrótu nie pomoże wiele, bo żeby dało się zweryfikować znaki wpisane przez użytkownika, hasło musi być hashowane w kilkuznakowych kawałkach (dla każdej możliwej odmiany maski). Lepszym rozwiązaniem w tym przypadku jest szyfrowanie haseł[4], korzystając z zewnętrznego, trudnego do zdobycia sekretu. A idealnie, w przypadku bardzo wrażliwych systemów, jak banki, skorzystanie z dedykowanego bezpiecznego urządzenia (tzw. HSM, Hardware Security Module) do weryfikacji haseł.

PRZYKŁAD 2: ATAKI KRYPTOGRAFICZNE

Czy SHA-1 jest bezpieczną funkcją skrótu? Oczywiście, od 2005 roku (kiedy zostały znalezione pierwsze podatności) jedyna poprawna odpowiedź na to pytanie brzmi „nie". A od 2017 roku (pierwsza znaleziona kolizja) nawet najwięksi sceptycy nie mogą mieć wątpliwości, że SHA-1 nie spełnia stawianych przed nią oczekiwań. Dokładnie rzecz ujmując, opublikowany atak pozwala organizacji z odpowiednio dużym budżetem wygenerować dwa pliki mające ten sam skrót SHA-1. Jak wielkim jest to problemem tak naprawdę? Rozważmy poniższe scenariusze:

Certyfikaty X.509 – w tym przypadku urząd certyfikacji podpisuje dane kontrolowane przez klienta, gwarantując swoim autorytetem, że dane na certyfikacie są zgodne ze stanem faktycznym. Mniej więcej w ten sposób działa obecnie weryfikacja certyfikatów na stronach internetowych – klient łączący się ze stroną HTTPS sprawdza, czy jej certyfikat jest podpisany przez któryś z zaufanych certyfikatów CA. Ale tutaj właśnie dotkliwie atakują nas nowe podatności. Potencjalny atakujący może wygenerować dwa certyfikaty – poprawny oraz złośliwy – z tym samym skrótem, a następnie poprosić urząd certyfikacji o podpisanie pierwszego z nich. Jeśli do weryfikacji integralności zostanie użyta funkcja SHA-1, podpis będzie ważny dla obu certyfikatów i atakujący niesłusznie otrzyma prawa do przedstawiania się jako druga domena.

Weryfikacja hasła – trzymanie hasła jako skrótu SHA-1 to nigdy nie był dobry pomysł – są od tego specjalne konstrukcje[5]. Ale, co ciekawe, nowe ataki nie wpływają w żaden sposób na bezpieczeństwo tak przechowywanych haseł. SHA-1, owszem, jest złamany zarówno w teorii, jak i w praktyce, ale tylko jeśli chodzi o odporność na kolizje (ang. *collision resistance*). Dalej (oficjalnie) nikt nie zna sposobu (lepszego niż metoda siłowa) na to, żeby ze skrótu hasła odzyskać oryginał (albo inną wiadomość z takim samym skrótem). To znaczy, że nawet w razie wycieku bazy haseł użytkownicy są (relatywnie) bezpieczni – o ile używają odpowiednio mocnych haseł.

Kody uwierzytelniające wiadomości (ang. *message authentication codes*) – ciągle jeszcze w praktyce można trafić na programy wykorzystujące `sha1(msg)` jako kryptograficzny podpis wiadomości msg. Tutaj również nowy atak nic nie zmienia... bo taki schemat jest i zawsze był niebezpieczny (istnieje możliwość przedłużenia wiadomości za pomocą ataków typu Hash Length Extension). Nawet wtedy, gdy SHA-1 jeszcze było „bezpieczne".

Identyfikacja kodu w systemie Git – tutaj trafiamy na grząski teren. Git wewnętrznie identyfikuje wszystkie commity po ich haszu – czyli zachodzi dokładnie sytuacja, kiedy podatności w SHA-1 komplikują nam życie. Z drugiej strony oryginalnie kryptograficzne bezpieczeństwo funkcji skrótu w Git-cie wcale nie było ważną własnością – SHA-1 było używane jako bardzo wiarygodna suma kontrolna. I rzeczywiście – w większości przypadków, jeśli atakujący może commitować do naszego repozytorium, to umiejętność produkcji dowolnych kolizji SHA-1 nie pomaga mu szczególnie we wrzuceniu złośliwego kodu do repozytorium (na przykład dlatego, że przy porównywaniu zawartości zawsze wygrywa ta „starsza") – argumentował tak wielokrotnie autor Gita, Linus Torvalds. Niestety z czasem powstały mechanizmy bezpieczeństwa opierające się na kryptograficznych własnościach skrótu (np. podpisywanie commitów), więc mimo wszystko protokół jest dziurawy i trwają pracę nad migracją na nowszy algorytm (konkretnie SHA-256).

ZAKOŃCZENIE

Czy hashowanie haseł jest konieczne? Czy SHA-1 jest łatwo złamać? Jak widać, wszystko zależy od okoliczności. Tym, dość ogólnym, artykułem chciałem zachęcić wszystkich do zastanowienia się, dlaczego niektóre rzeczy robimy tak jak robimy oraz czy zawsze typowe podejście jest aplikowalne do naszej sytuacji. Sprawdza się to oczywiście nie tylko w wąskiej dziedzinie bezpieczeństwa – ślepe podążanie za ogólnie przyjętą wiedzą (tzw. cargo cult) często prowadzi do nieoptymalnych rozwiązań.

JAROSŁAW JEDYNAK

msm@tailcall.net

Rozpoczynał karierę jako programista, wyspecjalizował się w security. Do niedawna jako analityk malware w CERT Polska zajmował się między innymi inżynierią wsteczną złośliwego oprogramowania i automatyzacją jego analizy. Obecnie pracuje jako security engineer w Google, spędza czas na reverse-engineeringu oraz prowadzi badania nad nowymi sposobami wykrywania złośliwego oprogramowania. W wolnym czasie programuje do szuflady, gra w konkursy CTF i ogląda koty w Internecie.

3. Swoją drogą, jest to możliwe – ale znacznie komplikuje implementacje.
4. Które z kolei w przypadku większości stron jest złym rozwiązaniem.
5. Tak zwane Password Based Key Derivation Functions – na przykład scrypt, bcrypt albo PBKDF2.