Uniwersytet w Białymstoku

Wydział Matematyki i Informatyki

Instytut Matematyki

Marcin Roszkowski

Bezpieczeństwo transmisji danych oraz uwierzytelnianie serwerów z wykorzystaniem protokołów SSL i TLS

Praca magisterska napisana
pod kierunkiem
dr hab. Anna Gomolińska, prof. UwB (promotor)
dr Mariusz Żynel (promotor pomocniczy)

Białystok 2018

Spis treści

Spis rysunków

Wstęp

Żyjąc w dobie nieustannie rozrastającej się sieci Internet, jesteśmy zarówno świadkami, jak i uczestnikami ciągłej wymiany danych. Stale zwiększająca się ilość urządzeń oraz użytkowników tworzących sieć, umożliwia im łatwiejszy dostęp i wykorzystanie usług takich jak SMTP, FTP oraz zdecydowanie najczęściej i najintensywniej eksploatowanej – usługi HTTP. Nad bezpieczeństwem wymienianych w ramach komunikacji danych, czuwają protokoły **SSL** (Secure Socket Layer) oraz **TLS** (Transport Layer Security).

Celem mojej pracy było scharakteryzowanie protokołów SSL oraz TLS, opis infrastruktury klucza publicznego, przedstawienie wektorów ataków wymierzonych w infrastrukturę klucza publicznego oraz omówienie problemów związanych z bezpieczeństwem stricte protokołu HTTP. Ponadto celem niniejszej pracy było podjęcie działań zwiększających poziom bezpieczeństwa podstawowych usług uruchomionych na serwerze obsługującym domenę *math.uwb.edu.pl* oraz opracowanie aplikacji webowej umożliwiającej weryfikację prawidłowości zabezpieczeń typowych usług internetowych na serwerach.

Część teoretyczną niniejszej pracy stanowiły zatem rozdziały: pierwszy, drugi, trzeci oraz czwarty. Rozdział pierwszy stanowi wstęp do kryptografii, w którym zostały opisane podstawowe jej elementy. W rozdziale tym, zostały również przedstawione protokoły SSL oraz TLS, rys historyczny oraz cele realizowane przez dane protokoły. W rozdziale drugim skupiłem się na przedstawieniu i opisaniu infrastruktury klucza publicznego. Rozdziały trzeci oraz czwarty zostały poświęcone przedstawieniu i naszkicowaniu problemów związanych z bezpieczeństwem protokołów oraz ich implementacji w bibliotekach i programach. Trzeci rozdział stanowi historyczny przegląd wektorów ataków wymierzonych w infrastrukturę klucza publicznego oraz urzędy certyfikacji. Przedstawione zostały w nim ataki takie jak, między innymi, Flame czy POODLE oraz incydenty dotyczące znanych urzędów certyfikacyjnych. Czwarty rozdział skupia się na relacji pomiędzy protokołami HTTP oraz SSL, TLS. Stanowi on również opis problemów wynikających z nieustannego rozrastania się oraz rozwoju sieci Internet.

Praktyczną część pracy opisuje rozdział piąty. Przedstawiony w nim został złożony proces aktualizacji oprogramowania serwera **math**, jego realizacja oraz napotkane problemy. Aktualizowane na serwerze **math** oprogramowanie to: *OpenSSL*, *OpenSSH*, serwer Apache 2 z nowym modułem PHP oraz Cy-

rus IMAP. Każda sekcja tego rozdziału zawiera istotne fragmenty napisanych skryptów, plików konfiguracyjnych oraz komend z odpowiednimi parametrami wykorzystanymi w procesie aktualizacji. Na rzecz części praktycznej pracy, stworzona została prosta aplikacja webowa umożliwiająca proste i sprawne zweryfikowanie połączeń SSL/TLS z typowymi usługami. Wykorzystane zostało również oprogramowanie *OpenSSL* umożliwiające sprawdzenie certyfikatu oraz protokołu. Ponadto, wykorzystane oprogramowanie *nmap* umożliwia sprawdzenie obsługiwanych algorytmów szyfrujących oraz ich poziomu bezpieczeństwa.

Rozdział 1

Protokoły SSL i TLS

W procesie projektowania sieci Internet, małą uwagę przykładano do bezpieczeństwa wymienianych poprzez nią danych, dużą nadzieję pokładając w uczciwości stron zaangażowanych w komunikację. Takie lekkomyślne podejście mogło zdawać egzamin w czasach, kiedy sieć Internet stanowiła niewielka liczba jednostek (głównie uniwersytetów), które nie wykazywały ani chęci, ani potrzeby nadużywania słabości i wykorzystywania podatności mechanizmu komunikacji. Obecnie, w momencie kiedy sieć Internet rozrosła się do niewyobrażalnie wielkich rozmiarów i w komunikację angażuje nieporównywalnie większą liczbę użytkowników, takie podejście nie ma racji bytu. W celu zamaskowania niedoskonałości protokołów, na których oparta jest komunikacja w sieci Internet, zaprojektowane zostały protokoły SSL oraz TLS. Protokoły SSL (Secure Socket Layer) oraz TLS (Transport Layer Security) są protokołami kryptograficznymi, zaprojektowanymi w celu zapewnienia bezpieczeństwa komunikacji odbywającej się poprzez niezabezpieczoną infrastrukturę – sieć Internet. Oparta jest ona na protokołach IP oraz TCP, które wykorzystywane są do opakowywania danych w małe pakiety gotowe do transportu. Pakiety te przebywają długą drogę, gdyż zanim dotrą do adresata, przekazywane są między wieloma systemami komputerowymi rozmieszczonymi w różnych miejscach na świecie. Na każdym etapie pokonywanej przez pakiet drogi, jest on narażony na przechwycenie i zmodyfikowanie przez osobę, która ma dostęp do komunikacji między stronami, w sposób niezauważony. W przypadku zaszyfrowanej komunikacji, taka osoba nadal będzie w stanie przechwycić wymieniane przez obie strony informacje, aczkolwiek nie będzie w stanie ich zmienić ani odszyfrować.

SSL/TLS w modelu OSI

W celu zgłębienia wiedzy na temat miejsca, w które najbardziej wpasowują się protokoły SSL/TLS, należy przyjrzeć się koncepcyjnemu modelowi OSI opisującemu strukturę komunikacji sieciowej. Model OSI składa się z 7 warstw, przedstawia je rysunek 1.1:

#	OSI Layer	Description	Example protocols
7	Application	Application data	HTTP, SMTP, IMAP
6	Presentation	Data representation, conversion, encryption	SSL/TLS
5	Session	Management of multiple connections	-
4	Transport	Reliable delivery of packets and streams	TCP, UDP
3	Network	Routing and delivery of datagrams between network nodes	IP, IPSec
2	Data link	Reliable local data connection (LAN)	Ethernet
1	Physical	Direct physical data connection (cables)	CAT5

Rysunek 1.1: Warstwy modelu OSI [1]

Organizacja komunikacji w ten sposób, zapewnia całkowitą rozłączność interesów. Protokoły wyższych warstw nie są zależne od sposobu implementacji funkcjonalności warstw niższych. Co więcej, protokoły na różnych warstwach mogą być usuwane lub dodawane oraz protokoły niższych warstw mogą być wykorzystywane przez protokoły z warstw wyższych. Bardzo dobrym przykładem zastosowania danych zasad w praktyce są protokoły SSL/TLS. Protokoły te umiejscowione są pomiędzy protokołem TCP a protokołami wyższych warstw – takimi jak HTTP. W przypadku, kiedy szyfrowanie nie jest wymagane, protokół TLS może zostać usunięty z modelu komunikacji, co nie zmieni działania protokołów warstw wyższych, które będą kontynuowały współpracę bezpośrednio z protokołem TCP. W przeciwnym przypadku, gdy szyfrowanie jest wymagane, można je wykorzystać do zaszyfrowania komunikacji protokołów takich jak HTTP oraz innych opartych na TCP, na przykład: SMTP, IMAP czy innych. Zatem w sytuacji, kiedy protokoły SSL/TLS są odpowiednio wdrożone, otwierając nowy kanał komunikacyjny skierowany do konkretnej usługi internetowej, można być relatywnie pewnym, że będziemy komunikować się z odpowiednim serwerem, a wysyłane dane nie zostaną przez nikogo przechwycone i dotrą one do serwera w stanie nienaruszonym. Wyróżnić zatem możemy cztery główne cele realizowane przez protokoły SSL/TLS:

Bezpieczeństwo kryptograficzne

Zapewnia bezpieczną komunikację i wymianę informacji pomiędzy dwiema stronami.

Interoperacyjność

Umożliwia niezależnym programistom wykorzystywanie wspólnych parametrów kryptograficznych w tworzonych przez nich programach i bibliotekach.

Rozszerzalność

Celem protokołu TLS jest, by ten nie był zależny od bieżących niskopoziomowych prymitywów kryptograficznych. Tym samym, umożliwiając ewentualną

migrację z jednego prymitywu kryptograficznego na drugi, bez potrzeby tworzenia nowego protokołu, który mógłby nieść ze sobą nowe podatności.

Wydajność

Ostatni, ale nie mniej istotny cel realizowany przez protokoły SSL/TLS. Zakłada, że wszystkie wymienione uprzednio cele osiągane będą przy akceptowalnym koszcie wydajności, redukując kosztowne operacje kryptograficzne do minimum, zapewniając schemat buforowania sesji, w celu nie wykonywania ich przy kolejnych połączeniach.

Historia protokołów

Protokół SSL został zaprojektowany przez firmę Netscape w czasach, kiedy przeglądarka Netscape Navigator dominowała w Internecie. SSL v1 nigdy nie ujrzał światła dziennego, dopiero druga wersja protokołu – SSL v2 – została opublikowana w listopadzie 1994 roku. Pierwsze wdrożenie protokołu zostało odnotowane w marcu 1995 roku. SSL w wersji 2.0 został wykorzystany w przeglądarce Netscape Navigator 1.1. W związku z licznymi podatnościami, protokół SSL v2 musiał zostać wycofany, co zmusiło firmę Netscape do rozpoczęcia prac nad kolejną wersją protokołu, która została opublikowana w listopadzie 1995 roku. Trzecia wersja protokołu SSL ustaliła model, który znany i używany jest do dziś. W styczniu 1999 roku opublikowany został protokół TLS v1.0, który w niewielkim stopniu różnił się od protokołu SSL v3. Kolejna wersja protokołu TLS, została wydana ponad 7 lat później, w kwietniu 2006 roku. TLS v1.1 zawierał głównie poprawki bezpieczeństwa, natomiast znaczne zmiany dotyczące protokołu zostały opublikowane w czerwcu 2003 roku – udokumentowane w specyfikacji RFC 3546[27]. W sierpniu 2008 roku wydana została kolejna wersja protokołu TLS – TLS v1.2 – dodana w niej została obsługa szyfrowania uwierzytelnionego natomiast ze specyfikacji usunięte zostały wszystkie wprowadzone na stałe prymitywy bezpieczeństwa, dzięki czemu protokół stał się w pełni elastyczny. Marzec 2011 roku, był miesiącem, w którym w dokumencie RFC 6176[22] oficjalnie uznano protokół SSL v2 za zdeprecjonowany. Kilka lat później w czerwcu 2015 roku los protokołu SSL v2 podzielił jego następca – SSL v3 – który w dokumencie RFC 7568[11] również został oficjalnie uznany za zdeprecjonowany. Prace nad następną wersją protokołu TLS – zatem TLS v1.3 – rozpoczęto w sierpniu 2013 roku. Kolejna wersja ma wnieść istotne zmiany mające na celu uproszczenie modelu, usunięcie podatności i mniej pożądanych funkcji oraz poprawę wydajności.

1.1 Kryptografia

Kryptografia jest nauką i sztuką bezpiecznej komunikacji. Zajmuje się ona badaniem technik bezpiecznej komunikacji odbywającej się w obecności osób

trzecich. Mimo iż kojarzona jest z czasami nowoczesnymi, swój początek miała tysiące lat temu, kiedy odnotowano pierwszą wzmiankę o greckiej metodzie szyfrowania *Skytale*. Kryptografia, którą znamy z mniej odległych czasów, swoje początki datuje na początek XX wieku. W tym okresie powstało bowiem kilkanaście mechanicznych urządzeń szyfrująco–deszyfrujących, służących do celów militarnych.

W przypadku, gdy kryptografia jest poprawnie wdrożona, odnosi się ona do trzech głównych wymagań bezpieczeństwa:

- poufności
- autentyczności
- integralności

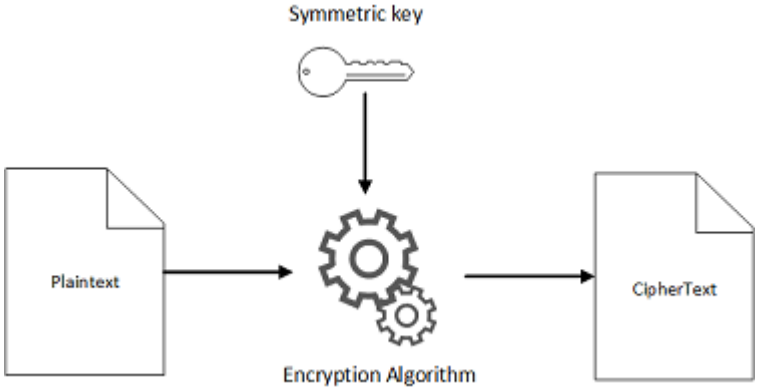
Na spełnienie powyższych wymagań pracuje wiele podstawowych elementów składowych kryptografii.

Prymitywy kryptograficzne

Na jej najniższym poziomie, kryptografia opiera się na różnych prymitywach kryptograficznych. Każdy z nich zaprojektowany jest w celu spełniania i realizowania konkretnych funkcjonalności, takich jak: szyfrowanie czy weryfikacja integralności. Prymitywy mogą być łączone w schematy bądź protokoły. W przypadku, kiedy występują w takiej postaci, są one najbardziej użyteczne, w przeciwieństwie do sytuacji, gdy występują one osobno.

Szyfrowanie symetryczne

Szyfrowanie symetryczne – zwane również kryptografią z wykorzystaniem klucza prywatnego – jest metodą szyfrowania wykorzystującą ten sam prywatny klucz do szyfrowania i odszyfrowywania danych. Pozwala ona na relatywnie bezpieczną wymianę danych poprzez niezabezpieczony kanał komunikacyjny.



Rysunek 1.2: Szyfrowanie symetryczne – szyfrowanie wiadomości [1]

Rysunek 1.2 ilustruje sytuację, w której za pomocą algorytmu szyfrującego i ustalonego przez obie strony klucza prywatnego, dokonuje się szyfrowanie informacji przez jedną ze stron. Adresat, który jest w posiadaniu tego samego klucza, może odszyfrować przesyłane przez nadawcę informacje, wykorzystując do tego uzgodniony klucz, który w tym przypadku będzie pełnił rolę deszyfratora. Komunikacja wykorzystująca szyfrowanie symetryczne jest bezpieczną metodą do momentu, w którym osoba postronna posiadająca dostępu do wymienianych przez obie strony (zaszyfrowanych) informacji, nie zgadnie bądź nie złamie klucza prywatnego.

Dobry algorytm kryptograficzny, to taki, w którego wyniku pracy otrzymujemy pozornie losowy szyfrogram. Po przeanalizowaniu go przez osobę postronną, mającą na celu złamanie szyfru, nie ujawnia on żadnych informacji na temat zaszyfrowanego tekstu. Dobrym przykładem przeciwnej sytuacji jest szyfr podstawieniowy. Na podstawie szyfrogramu otrzymanego w wyniku działania tego szyfru, można określić częstotliwość występowania liter w szyfrogramie i porównać ją z częstotliwością występowania liter w danym języku. Na podstawie takich obserwacji, osoba postronna jest w stanie odszyfrować szyfrogram, na przykład przy użyciu metody wyszukiwania wyczerpującego.

Szyfry strumieniowe

Szyfry strumieniowe stanowią metodę szyfrowania tekstu wykorzystującą klucz i algorytm szyfrujący do zaszyfrowania każdej kolejnej liczby binarnej nieskończonego strumienia pozornie losowych danych, wytworzonych przez rdzeń szyfru strumieniowego. W celu zaszyfrowania informacji, jeden bajt strumienia danych łączony jest z jednym bajtem danej informacji przy użyciu logicznej operacji XOR. Odwracalność wykonania operacji XOR pozwala na odszyfrowanie wiadomości. W tym celu, jeden bajt zaszyfrowanej wiadomości łączy się z odpowiadającym mu bajtem strumienia za pomocą operacji XOR. Proces szyfrowania pozostaje bezpieczny do momentu, w którym osoba postronna jest w stanie przewidzieć pozycję bajtów nieskończonego strumienia danych.

Szyfry blokowe

Szyfry blokowe są funkcjami transformacji, które kodują otrzymaną wiadomość w całości. Odróżnia to je zatem od szyfrów strumieniowych, które każdy bajt danej wiadomości szyfrują oddzielnie. W wyniku działania szyfru blokowego będziemy uzyskiwali ten sam szyfrogram dla różnych wiadomości przekazanych do szyfru do momentu, w którym klucz szyfrujący zostanie zmieniony. W związku z tą cechą, szyfry blokowe są szyframi deterministycznymi. Kluczową właściwością tego typu szyfrów jest to, że nieznaczna zmiana wiadomości wejściowej powoduje znaczne zmiany w otrzymanym szyfrogramie. Najpowszechniej używanym szyfrem blokowym jest *AES – Advanced Encryption Standard*.

Funkcje skrótu

Funkcje skrótu są algorytmami przekształcającymi przekazane na wejściu informacje dowolnej długości w losowy ciąg znaków o stałej długości – tzw. *hash*. Stricte kryptograficzne funkcje skrótu cechują się dodatkowo tym, iż na podstawie hasha powstałego w wyniku działania algorytmu niewykonalne obliczeniowo jest znalezienie wiadomości, na podstawie której dany hash powstał. Mając wiadomość oraz hash, niewykonalnym obliczeniowo jest znalezienie innej wiadomości, której odpowiada taki sam hash. Ponadto, niewykonalnym obliczeniowo jest również znalezienie dwóch wiadomości, którym odpowiada taki sam hash.

Funkcje skrótu są bardzo powszechnie używane w celu reprezentacji oraz porównywania dużych ilości danych, w związku z czym często bywają określane jako *fingerprints* (z ang. „odciski palca").

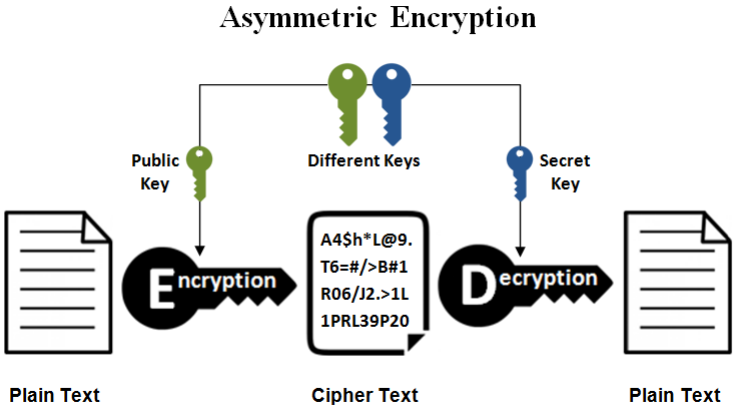
Najczęściej używaną funkcją skrótu jest *SHA1*, która generuje hash o długości 160 bitów, aczkolwiek zaleca się używanie silniejszego wariantu tej funkcji, zatem *SHA256*. W odróżnieniu do szyfrów, siła hasha nie jest określana na podstawie jego długości.

Message Authentication Codes – kod uwierzytelnienia wiadomości

Funkcje skrótu mogą być wykorzystywane w celu weryfikacji integralności danych pod warunkiem, że dane oraz hash transportowane są rozłącznie. W przypadku niespełnienia tego warunku, osoba posiadająca dostęp do tych zasobów może je zmodyfikować pozostając niezauważoną. MAC jest funkcją kryptograficzną, która rozszerza funkcję skrótu o uwierzytelnianie. Umożliwia ona utworzenie prawidłowego kodu uwierzytelnienia wiadomości jedynie osobie posiadającej klucz funkcji skrótu. Zatem w przypadku przechwycenia i zmodyfikowania wiadomości, adresat na podstawie MAC mógłby orzec, że otrzymana przez niego wiadomość nie jest tą, która została do niego wysłana.

Szyfrowanie asymetryczne

Metoda szyfrowania wykorzystująca zarówno klucz publiczny, jak i prywatny. Klucz publiczny udostępniany jest dla wszystkich zainteresowanych, natomiast klucz prywatny pozostaje znany tylko jego właścicielowi. Matematyczna zależność istniejąca pomiędzy tymi kluczami, umożliwia wykorzystanie ich użytecznych właściwości. W przypadku, w którym wiadomość zaszyfrowana zostanie kluczem publicznym strony A, może ona zostać odszyfrowana jedynie odpowiadającym mu kluczem prywatnym. W innym przypadku, wiadomość zaszyfrowana kluczem prywatnym strony A może zostać odszyfrowana przez każdego zainteresowanego, posiadającego udostępniony klucz publiczny tejże strony. Opisane sytuacje przedstawia rysunek 1.3.



Rysunek 1.3: Szyfrowanie asymetryczne – szyfrowanie i deszyfrowanie wiadomości [4]

Mimo interesujących właściwości, szyfrowanie asymetryczne nie jest odpowiednim rozwiązaniem, które można byłoby stosować w przypadku kodowania dużej ilości danych. W związku z tym, najczęściej stosowane jest do realizacji uwierzytelniania i negocjacji współdzielonych kluczy, wykorzystywanych do szyfrowania symetrycznego. Aktualnie, najpopularniejszym algorytmem szyfrowania asymetrycznego jest *RSA*.

Podpisy cyfrowe

Podpis cyfrowy jest schematem kryptograficznym, który umożliwia weryfikację autentyczności dokumentu, bądź wiadomości w postaci cyfrowej. Na podstawie algorytmu szyfrowania asymetrycznego (na przykład – *RSA*), generuje dwa powiązane ze sobą klucze. W celu utworzenia podpisu cyfrowego, oprogramowanie takie jak *Thunderbird* czy *MS Outlook* (klient poczty), generuje hash na podstawie danych, które mają zostać podpisane. Dany hash jest szyfrowany za pomocą klucza prywatnego. Razem z innymi danymi – takimi jak informacje o algorytmie szyfrującym, hash stanowi podpis cyfrowy. Może być on stosowany do różnego typu wiadomości – zaszyfrowanych, bądź w zwykłej, odkrytej formie [12].

Protokoły

Osobno, prymitywy kryptograficzne takie jak szyfrowanie czy funkcje skrótu przydatne są sporadycznie. Łączone w schematy lub protokoły, są w stanie spełniać złożone wymagania bezpieczeństwa.

Rozważmy sytuację, w której chcemy stworzyć prosty protokół umożliwiający bezpieczną wymianę informacji. Powinien on spełniać trzy główne wymagania bezpieczeństwa, zatem:

- poufność,

- integralność,
- poświadczenie o autentyczności.

W związku z tym iż protokół zapewnia wymianę dowolnej ilości wiadomości, w celu szyfrowania wymienianych informacji wykorzystany w nim zostanie algorytm szyfrowania symetrycznego (na przykład – *AES*). Ten element bezpieczeństwa gwarantuje poufność wymienianych danych, gdyż osoba postronna nie jest w stanie ich odczytać nie mając autoryzacji. Nie zapewnia on jednak kolejnego wymagania – integralności. Mimo, iż wymieniane przez obie strony wiadomości są zaszyfrowane, są one narażone na nieautoryzowaną modyfikację. W celu zagwarantowania integralności, należy użyć prymitywu *MAC* zatem – algorytmu uwierzytelnienia wiadomości. Do każdej wysłanej przez jedną ze stron zaszyfrowanych wiadomości, dołączony dodatkowo będzie kod uwierzytelnienia wygenerowany na podstawie wiadomości i klucza hashującego, znanego tylko przez obie strony biorące udział w komunikacji. W momencie, w którym te elementy bezpieczeństwa są zapewnione, osoba postronna w celu utrudnienia komunikacji między dwiema stronami, może oprócz niedopuszczenia do niej, zaszkodzić przechwytując zaszyfrowane wiadomości bądź wysyłając je kilkukrotnie. W celu uniknięcia danych sytuacji, protokół zostanie rozszerzony o funkcjonalność, która realizowała będzie przypisywanie numeru sekwencji do każdej wiadomości. Sprawdzenie tego numeru będzie częścią weryfikacji kodu uwierzytelnienia. W przypadku, w którym wykryta zostanie luka w numerach sekwencji, osoby biorące udział w komunikacji będą miały pewność tego, że wiadomość została przechwycona, natomiast w przypadku, w którym numer sekwencji zostanie powielony, będzie to informacją o tym, że wiadomość została zduplikowana. Istotnym elementem protokołu jest również ustalona pomiędzy stronami wiadomość, mająca na celu uzgodnienie i wskazanie momentu zakończenia komunikacji. Brak takiej wiadomości, umożliwia osobie postronnej dokonania tego, w nieoczekiwanym dla obu stron biorących udział w komunikacji momencie.

Mimo zapewnienia poufności i integralności wymienianych informacji, protokół w obecnej postaci nie umożliwia bezpiecznej negocjacji dwóch potrzebnych do rozpoczęcia komunikacji kluczy. W tym celu wykorzystany zostanie algorytm szyfrowania asymetrycznego, który umożliwi uwierzytelnienie obydwu stron, a następnie wykorzystanie schematu wymiany kluczy.

Ostatecznie, na podstawie tak skonstruowanego protokołu, obie strony uwierzytelniwszy się na początku komunikacji, wymieniają się potrzebnymi kluczami, zapewniającymi integralność i poufność wymienianych informacji, by w odpowiednim momencie wysłać ustaloną sekwencję kończącą proces komunikacji. Na wysokim poziomie abstrakcji, protokół skonstruowany w taki sposób, działaniem przypomina protokoły SSL oraz TLS. [1, Rozdz. 1]

1.2 Handshake

Handshake stanowi najbardziej skomplikowany element protokołu TLS. Podczas fazy handshake'u, dokonuje się negocjacja parametrów połączenia oraz uwierzytelnienia obydwu stron zainteresowanych rozpoczęciem bezpiecznej komunikacji. W ramach tego procesu, wymienianych jest od sześciu do dziesięciu wiadomości – ilość wymienionych wiadomości zależna jest od użytych funkcji. Handshake może występować w kilku odmianach, w zależności od konfiguracji i obsługiwanych rozszerzeń protokołu. W praktyce, występują trzy możliwe przypadki:

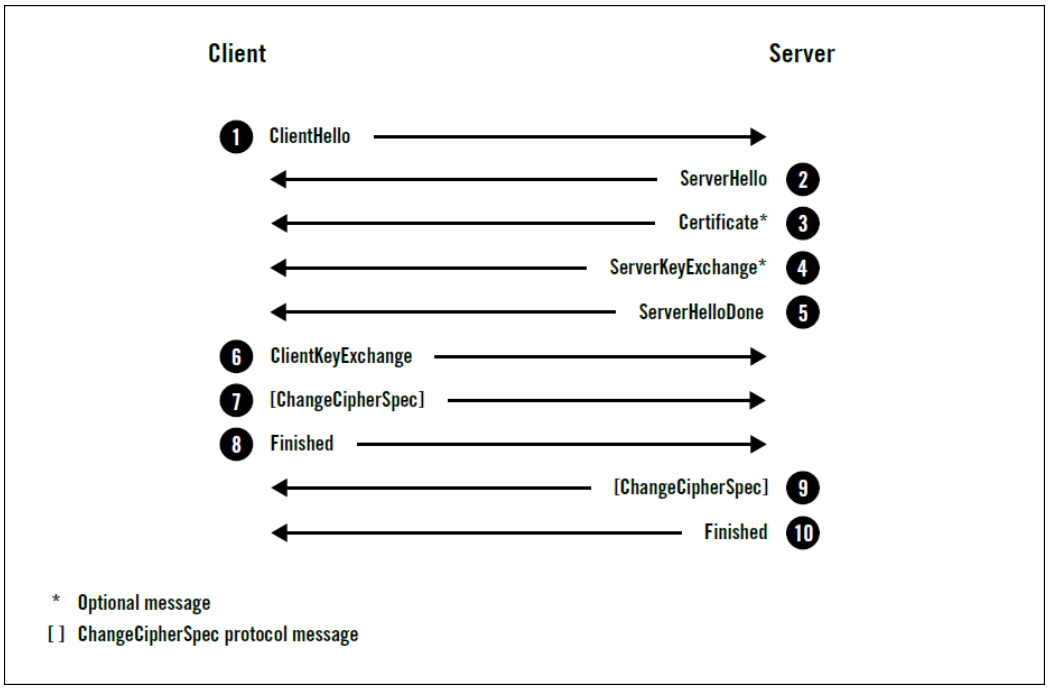
1. kompletny handshake z uwierzytelnieniem serwera,
2. skrócony handshake wznawiający wcześniej rozpoczętą sesję,
3. handshake z uwierzytelnieniem klienta i serwera.

1.2.1 Kompletny handshake z uwierzytelnieniem serwera

Każde połączenie nawiązane przy użyciu protokołu TLS zaczyna się wstępną wymianą informacji mającą na celu ustalenie wspólnych parametrów transmisji danych. W przypadku, kiedy klient ówcześnie nie nawiązał sesji z serwerem, obie strony wykonają pełen handshake w celu negocjacji sesji TLS. Podczas tego procesu, przez obie strony wykonane zostaną cztery główne aktywności:

1. Wymiana informacji na temat posiadanych możliwości.
2. Walidacja prezentowanego certyfikatu (bądź certyfikatów), lub uwierzytelnienie innymi środkami.
3. Ustalenie współdzielonego wstępnego klucza sesji.
4. Weryfikacja, czy wymieniane w ramach handshake'u wiadomości nie były modyfikowane przez osoby postronne.

Schemat 1.4 przedstawia najpowszechniejszy przypadek handshake'u TLS. Jest to przypadek, w którym uwierzytelniany jest tylko serwer.



Rysunek 1.4: Schemat kompletnego handshake'u z uwierzytelnieniem serwera [1]

1. Klient rozpoczyna nowy handshake i wysyła informacje o posiadanych możliwościach serwerowi.
2. Na podstawie możliwości swoich i klienta, serwer decyduje o optymalnych parametrach połączenia.
3. W przypadku, w którym uwierzytelnienie serwera jest wymagane, wysyła on swój certyfikat.
4. W zależności od wybranego sposobu wymiany kluczy, serwer wysyła dodatkowe informacje wymagane do wygenerowania klucza sesji.
5. Serwer sygnalizuje wykonanie swoich obowiązków dotyczących negocjacji warunków połączenia.
6. Klient wysyła dodatkowe informacje wymagane do wygenerowania klucza sesji.
7. Klient informuje serwer, o uzyskaniu wszystkich potrzebnych informacji do nawiązania bezpiecznego połączenia.
8. Klient wysyła kody uwierzytelnienia wiadomości wymienianych w ramach handshake'u.

9. Serwer informuje klienta, o uzyskaniu wszystkich potrzebnych informacji do nawiązania bezpiecznego połączenia.

10. Serwer wysyła kody uwierzytelnienia wiadomości wymienianych w ramach handshake'u.

Wiadomość ClientHello

Wiadomość ClientHello jest zawsze pierwszą wiadomością wysłaną w ramach nowego handshake'u. Zawarte są w niej informacje określające możliwości i preferencje klienta odnośnie do sposobu oraz warunków nawiązania bezpiecznego połączenia. Wiadomość ta, może być również wysyłana w przypadku, gdy klient chce renegocjować warunki bezpiecznego połączenia, bądź w przypadku odpowiedzi na żądanie serwera o renegocjację. Rysunek 1.5 przedstawia uproszczony przykład wiadomości ClientHello.

```
Handshake protocol: ClientHello
    Version: TLS 1.2
    Random
        Client time: May 22, 2030 02:43:46 GMT
        Random bytes: b76b0e61829557eb4c611adfd2d36eb232dc1332fe29802e321ee871
    Session ID: (empty)
    Cipher Suites
        Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
        Suite: TLS_RSA_WITH_AES_128_GCM_SHA256
        Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
        Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA
        Suite: TLS_RSA_WITH_AES_128_CBC_SHA
        Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA
        Suite: TLS_RSA_WITH_RC4_128_SHA
    Compression methods
        Method: null
    Extensions
        Extension: server_name
            Hostname: www.feistyduck.com
        Extension: renegotiation_info
        Extension: elliptic_curves
            Named curve: secp256r1
            Named curve: secp384r1
        Extension: signature_algorithms
            Algorithm: sha1/rsa
            Algorithm: sha256/rsa
            Algorithm: sha1/ecdsa
            Algorithm: sha256/ecdsa
```

Rysunek 1.5: Uproszczony przykład **ClientHello** [1]

Protocol version

Pole wskazujące najlepszą wersję protokołu obsługiwaną przez klienta.

Random

Pole przechowujące 32 bajty danych, z których pierwsze 28 bajtów jest generowanych losowo. Pozostałe 4 bajty mogą stanowić dodatkową informację dotyczącą czasu, dostarczoną przez przeglądarkę klienta. Jest to informacja dotycząca właściwego czasu zapytania oraz zjawiska związanego z układami synchronicznymi układów cyfrowych[8]. W przypadku, gdy ta informacja nie zostanie dostarczona, 4 ostatnie bajty stanowią cztery losowo wybrane znaki alfanumeryczne. Możliwość zawarcia takiej informacji w wiadomości Random nie jest wymagana przez TLS, aczkolwiek protokoły wysopokoziomowe, bądź protokoły aplikacji mogą zawierać dodatkowe wymagania odnośnie do tej informacji. Zarówno klient jak i serwer przekazują sobie losowe wartości danych, co czyni każdy handshake unikalnym. Odgrywają one istotną rolę w uwierzytelnianiu zapobiegając przed atakami powtórzeniowymi oraz weryfikują integralność początkowej wymiany danych.

Session ID

Pole przechowujące informacje o identyfikatorze sesji. W przypadku pierwszego połączenia, pole jest puste, co oznacza, że klient nie chce wznowić połączenia. W przypadku kolejnych połączeń, pole może zawierać unikalny klucz sesji, który umożliwi serwerowi zlokalizowanie w pamięci podręcznej poprawnego jej stanu. Klucz sesji zazwyczaj składa się z 32 bajtowego ciągu losowo wybranych znaków alfanumerycznych. Osobno nie stanowi wartościowego zasobu w kontekście nawiązywania bezpiecznego połączenia.

Cipher suites

Blok odnoszący się do obsługiwanych przez klienta zestawów algorytmów szyfrujących, podanych w kolejności zgodnej z preferencjami klienta.

Compression

Pole przechowujące informacje o obsługiwanych metodach kompresji (domyślnie **null**, co oznacza brak obsługiwanych metod kompresji).

Extensions

Blok, w którym znajdują się informacje dotyczące dowolnej ilości rozszerzeń, które mogą dostarczać dodatkowych danych.

Wiadomość ServerHello

Wiadomość wysyłana przez serwer do klienta, zawierająca wybrane spośród proponowanych przez klienta parametry połączenia. Struktura wiadomości (przedstawiona na rysunku 1.6) przypomina strukturą wiadomość ClientHello, z zasadniczą różnicą iż zawiera tylko po jednej wartości dla każdego pola. W przypadku, kiedy serwer nie obsługuje wersji protokołu proponowanej przez klienta, w polu Version zawiera inną, proponowaną wersję protokołu.

```
Handshake protocol: ServerHello
    Version: TLS 1.2
    Random
        Server time: Mar 10, 2059 02:35:57 GMT
        Random bytes: 8469b09b480c1978182ce1b59290487609f41132312ca22aacaf5012
    Session ID: 4cae75c91cf5adf55f93c9fb5dd36d19903b1182029af3d527b7a42ef1c32c80
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    Compression method: null
    Extensions
        Extension: server_name
        Extension: renegotiation_info
```

Rysunek 1.6: Przykład **ServerHello** [1]

Certificate

Wiadomość zazwyczaj zawiera łańcuch X.509 certyfikatu serwera. Certyfikaty są wysyłane w odpowiedniej kolejności – główny certyfikat wysyłany jest jako pierwszy, kolejno wysyłane są certyfikaty pośredniczące. Wiadomość Certificate jest opcjonalna, ze względu na to iż nie wszystkie zestawy wykorzystują uwierzytelnianie oraz nie wszystkie metody uwierzytelniania wymagają certyfikatów.

ServerKeyExchange

Wiadomość zawierająca dodatkowe informacje wymagane do procesu wymiany kluczy. Zawartość wiadomości zależy od uzgodnionych zestawów algorytmów szyfrujących. W przypadkach, gdy serwer nie musi wysyłać takich informacji, wiadomość ServerKeyExchange nie jest wysyłana.

ServerHelloDone

Komunikat wysyłany do klienta, informujący go o tym, że serwer wysłał do niego wszystkie zamierzone wiadomości. Po wysłaniu tej informacji, serwer oczekuje na kolejne wiadomości od klienta.

ClientKeyExchange

Obowiązkowa wiadomość wysyłana przez klienta, jej zawartość zależy od uzgodnionego zestawu algorytmu szyfrującego.

ChangeCipherSpec

Wiadomość informująca o tym, że jej nadawca uzyskał wystarczającą ilość informacji w celu wytworzenia parametrów połączenia, wygenerowania kluczy szyfrujących oraz zgłoszenia gotowości do przejścia na szyfrowaną, bezpieczną komunikację. Istotną informacją jest iż ChangeCipherSpec nie jest bezpośrednio wiadomością wysyłaną w ramach handshake'u protokołu SSL/TLS.

Finished

Komunikat informujący o zakończeniu wstępnej wymiany informacji mającej na celu ustalenie wspólnych parametrów transmisji danych. W związku z tym iż zawartość wiadomości Finished jest zaszyfrowana, fakt ten umożliwia bezpieczną wymianę wymaganych danych w celu weryfikacji integralności całego procesu handshake'u. W protokole TLS 1.2, domyślna długość wiadomości Finished stanowi 12 bajtów, aczkolwiek zestawy algorytmów szyfrujących mogą wykorzystywać wiadomości o większym rozmiarze.

1.2.2 Skrócony handshake wznawiający wcześniej rozpoczętą sesję

Nawiązanie pełnego handshake'u jest procesem złożonym i wymagającym zazwyczaj intensywnego wykorzystania procesora. Procesu tego można uniknąć w przypadku nawiązywania kolejnego połączenia pomiędzy klientem oraz serwerem, zastosowując mechanizm Session Resumption (z ang. wznawiania sesji). Mechanizm ten obliguje klienta oraz serwer do przechowywania parametrów bezpieczeństwa sesji przez pewien określony czas, upływający od zakończenia połączenia, w ramach którego uzgodnione zostały warunki handshake'u. W przypadku, kiedy serwer chce umożliwić wykorzystanie mechanizm wznawiania sesji, przypisuje jej unikalny identyfikator, który wysyła klientowi w wiadomości ServerHello, W celu wykorzystania przez klienta mechanizmu i wznowienia wcześniej nawiązanej sesję, klient wysyła odpowiedni jej identyfikator w wiadomości ClientHello. Jeżeli serwer jest skłonny wznowić sesję, wysyła jej (ten sam) identyfikator w wiadomości ServerHello. Wiadomość ta zawiera również zestaw nowych kluczy, wygenerowanych z użyciem ustalonego w poprzedniej sesji tajnego klucza głównego (ang. master secret). Po wysłaniu tej wiadomości do klienta, serwer przełącza się na szyfrowaną komunikację, a następnie wysyła komunikat o zakończeniu procesu wznawiania sesji. W odpowiedzi, po otrzymaniu przez klienta wiadomości od serwera, przełącza się

on na szyfrowaną komunikację i wysyła komunikat o nawiązaniu bezpiecznego połączenia.

1.2.3 Handshake z uwierzytelnieniem klienta i serwera

Przypadek handshake'u, w którego ramach dokonuje się również uwierzytelnienia klienta. Jest on możliwy wyłącznie w momencie, gdy uwierzytelniony został już serwer. W celu wykorzystania mechanizmu handshake'u z uwierzytelnieniem obu stron, serwer wysyła do klienta wiadomość CertificateRequest.

CertificateRequest

Wiadomość zawierająca listę obsługiwanych typów certyfikatów oraz listę akceptowanych urzędów certyfikacyjnych.

CertificateVerify

Wysyłana przez klienta wiadomość, stanowiąca dowód tego iż posiada on odpowiadający kluczowi publicznemu zawartemu w wysłanym w ramach wiadomości CertificateRequest certyfikacie klucz prywatny. [1, Rozdz. 2, *Handshake Protocol*]

1.3 Wymiana kluczy

Celem wymiany kluczy jest wygenerowanie wstępnego tajnego klucza (z ang. premaster secret), na podstawie którego zostanie wygenerowany tajny klucz główny (z ang. master secret), od którego zależy bezpieczeństwo sesji protokołu TLS. Protokół ten obsługuje wiele algorytmów wymiany kluczy w celu obsługiwania różnych typów certyfikatów, asymetrycznych algorytmów szyfrowania oraz protokołów uzgadniania kluczy. W zależności od uzgodnionego przez obie strony zestawu, wykorzystywany jest odpowiedni algorytm wymiany kluczy. W praktyce, głównie wykorzystuje się cztery algorytmy: RSA, DHE_RSA, ECDHE_RSA oraz ECDHE_ECDSA.

RSA

Standardowy algorytm wymiany kluczy, którego prostota jest jego największą wadą. Ze względu na istotną podatność jest wypierany przez algorytmu obsługujące mechanizm utajnienia przekazywania. Dzięki RSA, klient dostarcza serwerowi zaszyfrowany jego kluczem publicznym wygenerowany przez klienta wstępny tajny klucz (z ang. premaster secret key).

DHE_RSA

Algorytm Diffiego–Hellmana jest dobrze skonstruowanym algorytmem, który obsługuje utajnienie przekazywania. DHE jest właściwie algorytmem uzgadniania klucza, który w protokole TLS wykorzystywany jest razem z algorytmem RSA w celu uzupełnienia go o uwierzytelnianie. W stosunku do algorytmu RSA jest on wolniejszy.

ECDHE_RSA, ECDHE_ECDSA

Algorytmy wykorzystujące krzywą eliptyczna Diffiego–Hellmana, oparte na relatywnie nowym typie kryptografii (z ang. elliptic curve cryptography). Obsługuje utajnianie przekazywania, przy czym jest stosunkowo szybszy od algorytmu DHE_RSA. W celu uwierzytelniania wykorzystują algorytm ECDSA, bądź jak poprzedni omawiany algorytm – RSA. [1, Rozdz. 2, *Key Exchange*]

1.4 Uwierzytelnianie

Uwierzytelnianie w protokole TLS jest ściśle związane z algorytmami wymiany kluczy. Powodem takiego zabiegu jest minimalizacja ilości wykonania kosztownych operacji kryptograficznych. W większości przypadków, podstawą uwierzytelniania będą obsługiwane przez certyfikaty asymetryczne algorytmy szyfrujące – zazwyczaj **RSA** lub **ECDSA**. W momencie, kiedy tożsamość certyfikatu jest potwierdzona, rozpoczyna się proces wymiany kluczy uzgodnioną metodą, w celu wykorzystania ich do uwierzytelnienia się przez obie strony.

Podczas wymiany kluczy metodą **RSA**, klient generuje losową wartość *premaster secret*, szyfruje ją kluczem publicznym serwera, a następnie wysyła. Serwer, będąc w posiadaniu klucza prywatnego odpowiadającego kluczowi publicznemu, odszyfrowuje przesłaną przez klienta wiadomość, w wyniku czego, otrzymuje odkodowaną wartość *premaster secret*. W przypadku tego algorytmu, uwierzytelnienie jest domniemane. Dzieje się tak, ze względu na to iż zakłada się, że tylko serwer, będąc w posiadaniu odpowiadającego kluczowi publicznemu klucza prywatnego jest w stanie odszyfrować *premaster secret*, by następnie wytworzyć poprawne klucze sesji oraz wiadomość **Finished**.

Wymiana kluczy z użyciem algorytmów **DHE** oraz **ECDHE**, obliguje serwer do zawarcia w wiadomości zaszyfrowanych kluczem prywatnym parametrów serwera, podczas trwania tego procesu. Dzięki takiemu zabiegowi, klient będą w posiadaniu klucza publicznego serwera, może odszyfrować wiadomość, by następnie zweryfikować, czy otrzymane w niej parametry dotyczą adekwatnego serwera. [1, Rozdz. 2, *Authentication*]

1.5 Szyfrowanie

Szyfrowanie w ramach protokołu TLS może odbywać się z wykorzystaniem takich szyfrów jak 3DES, AES, ARIA, CAMELLIA, RC4 lub SEED. Najpowszechniejszym i najczęściej wykorzystywanym z wymienionych szyfrów, jest AES. Obsługiwane są trzy typy szyfrowania: strumieniowe, blokowe oraz szyfrowanie uwierzytelnione. Walidacja integralności jest częścią procesu szyfrowania protokołu TLS i jest obsługiwana jawnie z poziomu protokołu, bądź niejawnie, przez ustalony szyfr.

Szyfry strumieniowe

Szyfrowanie, w przypadku wykorzystania szyfrów strumieniowych, dzieli się na dwa etapy. W pierwszym etapie, wyliczane są kody uwierzytelnienia numeru sekwencji rekordu, nagłówka oraz jawnego tekstu do zaszyfrowania. Dołączenie nagłówka do kodu uwierzytelnienia, zapewnia, że niezaszyfrowane w nagłówku dane, nie zostaną zmodyfikowane, natomiast dołączenie do niego również numeru sekwencji, zabezpiecza wiadomość przez jej zduplikowaniem. Drugi etap, stanowi zaszyfrowanie jawnego tekstu oraz kodu uwierzytelnienia, w celu utworzenia zakodowanej wiadomości.

Szyfry blokowe

W przypadku wykorzystania szyfrów blokowych, kodowanie jest nieco bardziej skomplikowane, ze względu na konieczność ominięcia właściwości szyfrowania blokowego. Proces składa się z następujących kroków:

1. Wyliczenie kodu uwierzytelnienia numeru sekwencji, nagłówka oraz jawnego tekstu.
2. Utworzenie *paddingu*[21], w celu zapewnienia tego, że długość danych przed szyfrowaniem jest wielokrotnością rozmiaru bloku szyfru.
3. Wygenerowanie niemożliwego do przewidzenia *wektora inicjującego*[16], o długości równej rozmiarowi bloku szyfru.
4. Wykorzystanie trybu wiązania bloków zaszyfrowanych (CBC[5]), w celu zaszyfrowania tekstu jawnego, kodu uwierzytelnienia oraz paddingu.
5. Wysłanie wektora inicjującego razem z zaszyfrowaną informacją.

Szyfrowanie uwierzytelnione

Szyfrowanie uwierzytelnione łączy w sobie walidację integralności oraz szyfrowanie. Szyfr ten może wydawać się skrzyżowaniem szyfrów strumieniowych oraz blokowych, aczkolwiek nie wykorzystuje on ani paddingu, ani wektora

inicjującego. Wykorzystuje on natomiast specjalną, unikalną wartość – *nonce* (wartość jednorazową). Metoda ta, jest aktualnie najbardziej preferowaną metodą szyfrowania z dostępnych metod protokołu TLS. Proces kodowania z wykorzystaniem szyfrowania uwierzytelnionego składa się z następujących kroków:

1. Wygenerowanie unikalnej, 64 bitowej wartości *nonce*.
2. Podczas szyfrowania jawnego tekstu z wykorzystaniem uwierzytelnionego algorytmu szyfrowania, jako dodatkowa informacja dostarczany jest do niego numer sekwencji oraz nagłówek rekordu, w celu walidacji integralności.
3. Wysłanie unikalnej wartości *nonce* razem z zaszyfrowaną informacją.

[1, Rozdz. 2, *Encryption*]

1.6 Renegocjacja

Rozpoczęcie wstępnej wymiany informacji mającej na celu ustalenie wspólnych parametrów transmisji danych, w obrębie istniejącej bezpiecznej sesji, określane jest terminem renegocjacji. Protokół umożliwia klientowi renegocjację warunków połączenia w każdym jej momencie. W tym celu, klient wysyła ponownie wiadomość **ClientHello**, tak jak na początku komunikacji. Taka sytuację można określić jako renegocjację zainicjowaną przez klienta. W przypadku, kiedy serwer chce renegocjować warunki połączenia, wysyła wiadomość **TLS HelloRequest**, sygnalizując klientowi, by ten zaprzestał wysyłania danych i zainicjował nowy handshake. Sytuacja ta, określana jest jako renegocjacja zainicjowana przez serwer. Tak zaprojektowany mechanizm renegocjacji, nie jest uznawany za bezpieczny. Ważną poprawkę bezpieczeństwa dotyczącą mechanizmu, opisuje dokument **RFC 5746**[28]. [1, Rozdz. 2, *Renegotiation*]

1.7 Zestawy algorytmów szyfrujących

Zestaw algorytmów szyfrujących stanowi zbiór wyselekcjonowanych prymitywów kryptograficznych oraz innych parametrów dokładnie definiujących sposób, w jaki wdrożone zostaną zabezpieczenia. W uogólnieniu, pakiet określają następujące atrybuty:

- Metoda uwierzytelniania
- Metoda wymiany kluczy
- Algorytm szyfrujący

- Rozmiar klucza szyfrowania
- Tryb szyfrowania (jeżeli dotyczy)
- Algorytm uwierzytelniania wiadomości (jeżeli dotyczy)
- Algorytm PRF[23] (tylko w TLS 1.2 – w przeciwnym razie, zależy od protokołu)
- Funkcja skrótu wykorzystana do wytworzenia wiadomości **Finished** (TLS 1.2)
- Długość struktury *verify_data* (TLS 1.2)

Nazwa zestawu algorytmów szyfrujących zazwyczaj jest spójna, aczkolwiek długa, gdyż składa się z nazw metod wymiany klucza, uwierzytelniania, nazwy algorytmu szyfrującego oraz opcjonalnie z nazwy algorytmu uwierzytelniania wiadomości i algorytmu PRF. Pozwala ona na sprawne rozpoznanie najistotniejszych wykorzystanych w zestawie algorytmów. W celu uzyskania dokładniejszych informacji, należy jednak zapoznać się ze specyfikacją RFC danego zestawu. [1, Rozdz. 2, *Cipher Suites*]

Rozdział 2

Infrastruktura klucza publicznego (PKI)

Kryptografia klucza publicznego umożliwia nawiązanie bezpiecznej komunikacji ze stronami, których klucze publiczne posiadamy. Niesie ona jednak ze sobą pewne problematyczne kwestie takie jak: w jaki sposób nawiązać komunikację ze stroną, której klucza nie posiadamy, jak przechowywać i przywoływać klucze publiczne oraz w jaki sposób, robić to w obecności milionów serwerów i miliardów ludzi. W celu rozwiązania zarówno tych oraz innych problemów została utworzona infrastruktura klucza publicznego. [1, Rozdz. 3]

2.1 Certyfikaty

Certyfikat jest dokumentem cyfrowym zawierającym klucz publiczny, informacje dotyczące podmiotu dla którego został on wystawiony oraz podpis cyfrowy organu wystawiającego certyfikat. W kontekście certyfikatów używa się terminu – *zaufany*. W odniesieniu do infrastruktury klucza publicznego, termin ten wykorzystywany jest jedynie w bardzo technicznym tego słowa znaczeniu. Przez zaufany certyfikat rozumie się iż jest on sprawdzony oraz zatwierdzony jako jeden z potwierdzonych certyfikatów przez odpowiedni organ.

2.2 Struktura certyfikatów

Certyfikat posiada prostą strukturę, gdyż składa się on z pól informacyjnych oraz ewentualnych rozszerzeń. Niektóre pola mogą zawierać inne struktury.

Wersja

Pole przechowujące wersję certyfikatu. Wyróżnia się trzy wersje, numerowane od 1–3, które zakodowane są jako wartości odpowiednio 0, 1 oraz 2. Certyfikat w wersji 1 obsługuje jedynie podstawowe pola, natomiast certyfikat w wersji 2

rozszerzony jest o dwa dodatkowe pola – unikalne identyfikatory. Wersja 3 jest najczęściej spotykaną wersją certyfikatu, zawiera ona dodatkowo rozszerzenia.

Numer seryjny

Początkowo, numer seryjny certyfikatu stanowił ciąg dodatnich liczb całkowitych, jednoznacznie identyfikujący wystawiony przez organ certyfikat. W celu utworzenia drugiej warstwy bezpieczeństwa, przyjęto dodatkowe wymagania – numery seryjne muszą być zatem nieprzewidywalnymi, niesekwencyjnymi numerami, zawierającymi co najmniej 20 bitów entropii[14].

Emitent

Złożone pole przechowujące informacje o wyróżniającej nazwie (*distinguished name (DN)*) organu wystawiającego certyfikat. Może zawierać wiele komponentów, w zależności od reprezentowanej jednostki.

Ważność

Stanowi informację dotyczącą ważności, przechowuje zatem datę rozpoczęcia oraz upływu ważności certyfikatu.

Podmiot

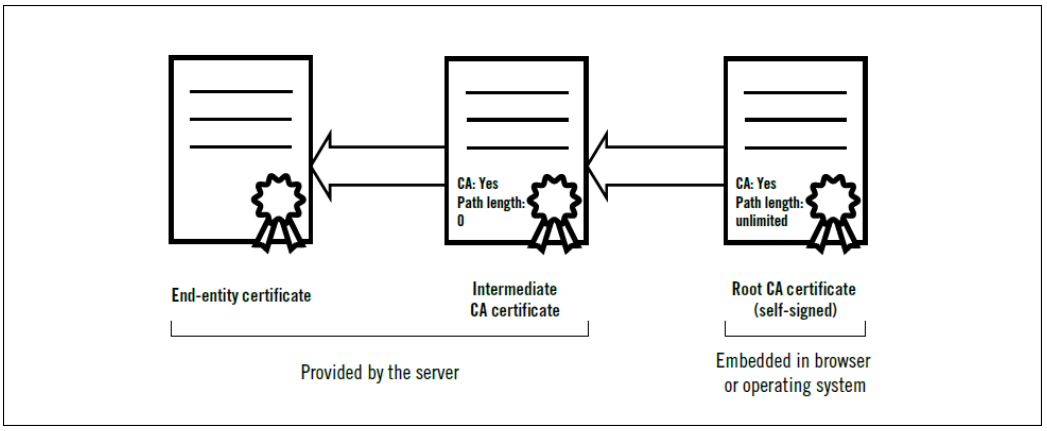
Pole stanowiące wyróżniającą oraz identyfikującą nazwę podmiotu, do którego przypisany jest klucz publiczny.

Klucz publiczny

Pole przechowujące informacje o kluczu publicznym w odpowiedniej strukturze – kolejno – identyfikator algorytmu, opcjonalne parametry oraz klucz publiczny. [1, Rozdz. 3, *Certificates*]

2.3 Łańcuchy certyfikatów

W celu pomyślnego potwierdzenia tożsamości, w większości przypadków niewystarczającym jest przedstawienie certyfikatu jedynie podmiotu docelowego. Ze względów technicznych, administracyjnych oraz bezpieczeństwa, używane są łańcuchy certyfikatów. Przykładowy schemat łańcucha certyfikatu przedstawia rysunek 2.1. Każdy serwer zobligowany jest do przedstawienia łańcucha certyfikatu prowadzącego do zaufanego urzędu certyfikacji.



Rysunek 2.1: Schemat łańcucha certyfikatu [1]

Z wielu względów, klucz urzędu certyfikacji stanowi bardzo istotną wartość, zarówno dla samego urzędu, jak i całego ekosystemu. Dzieje się tak ze względu na to iż sam klucz stanowi wysoką wartość finansową, natomiast starsze, szeroko rozdystrybuowane klucze, są efektywnie niezastąpione, ze względu na to iż wiele zbiorów głównych urzędów certyfikujących (*root stores*) nie jest już uaktualnianych. Ponadto, w przypadku kiedy klucz zostanie przechwycony, może zostać użyty do wystawienia fałszywych certyfikatów dla dowolnej domeny. Przechwycony klucz musiałby zostać unieważniony, co spowodowałoby unieważnienie wszystkich certyfikatów zależących od klucza tego urzędu certyfikacji. Istotnym wymaganiem wobec tych instytucji, jest, by klucz główny używany był jedynie poprzez wydanie bezpośredniego polecenia (zakazując jednocześnie automatyzacji tego procesu) oraz by był przechowywany w trybie offline.

Certyfikacja krzyżowa

Certyfikacja krzyżowa stanowi bardzo istotny element procesu certyfikacji. W celu sprawnego rozpowszechnienia nowych kluczy głównych, stosuje się praktykę, gdzie klucze te podpisywane są przez inne urzędy certyfikacji, dobrze znane i rozpowszechnione już w *root stores*. Takie podejście, umożliwi nowym kluczom bycie rozpoznawalnymi, by w przyszłości, nie musiały być zależne od innych.

Serwer powinien zapewnić tylko jeden łańcuch certyfikatu, aczkolwiek w praktyce, może istnieć wiele *ścieżek zaufania* (*trust paths*). Wynikiem zastosowania certyfikacji krzyżowej może być sytuacja, gdzie jedna ścieżka będzie prowadziła do głównego, natomiast inna do alternatywnego *CA*. Mogą występować sytuacje, w których urząd certyfikacji wystawia wiele certyfikatów dla tych samych kluczy. W przypadku zastosowania do podpisania certyfikatu dwóch algorytmów – np. SHA1 oraz SHA256 – urząd certyfikacji może użyć

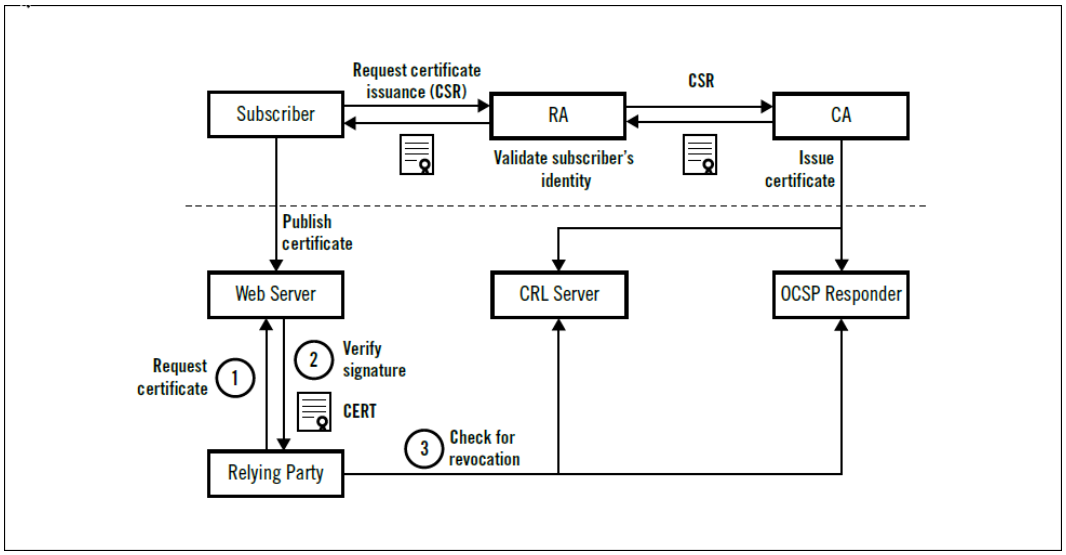
ponownie tego samego klucza wystawiając nowy certyfikat, którego ścieżka będzie inna.

Budowanie ścieżek zazwyczaj prowadzi do błędów i komplikacji. Spowodowane one mogą być czynnikiem ludzkim, bądź różnymi problemami związanymi z użytecznością. Według testów przeprowadzonych w kwietniu 2018 roku, szacuje się iż 2.2% witryn posiada błędną konfigurację łańcuchów certyfikatów. Z drugiej strony, budowanie ścieżek oraz ich walidacja jest przyczyną wielu problemów bezpieczeństwa związanych z oprogramowaniem klienckim. Biorąc pod uwagę niekompletne, niejasne oraz konkurencyjne standardy, nie jest to zaskakujące. Początkowo, wiele bibliotek walidujących nie spełniało poprawnie powierzonych im podstawowych zadań, takich jak potwierdzenie, czy wystawiony certyfikat przynależy do danego urzędu certyfikacji. Obecnie używane biblioteki są bezpieczne wyłącznie dlatego iż poprawione w nich zostały najpoważniejsze błędy i luki, a nie dlatego, że były bezpieczne od samego początku. [1, Rozdz. 3, *Certificate Chains*]

2.4 Urzędy certyfikacji

Urzędy certyfikacji (*Certificate Authorities* – CAs) stanowią najważniejszy element obecnego internetowego modelu zaufania. Zadaniem tego podmiotu jest wystawianie dowolnym domenom cyfrowych certyfikatów, poświadczających własność klucza publicznego przez posiadającego certyfikat. *CA* stanowi zaufaną trzecią stronę[29] w danym modelu, zarówno od strony właściciela certyfikatu, jak i od strony nań się powołującej. [1, Rozdz. 3, *Certificate Authorities*]

2.5 Cykl życia certyfikatu



Rysunek 2.2: Schemat cyklu życia łańcucha [1]

Na rysunku 2.2 przestawiającym schemat cyklu życia certyfikatu, możemy wyróżnić wiele podmiotów biorących w nim udział.

Podmiot końcowy

Mianem podmiotu końcowego określa się stronę, która chce zapewnić bezpieczeństwo udzielanych usług. Zatem jest to strona wnioskująca o certyfikat.

Urząd rejestracji

Urząd rejestracji wykonuje określone funkcje z zakresu zarządzania, będące związane z wystawieniem certyfikatu. Przykładowo, urząd ten może mieć za zadanie potwierdzenie tożsamości podmiotu końcowego, przed bezpośrednim żądaniem wydania certyfikatu. W praktyce, wiele urzędów certyfikacji pełni również rolę urzędu rejestracji.

Urząd certyfikacji

Zaufana strona wydająca certyfikat poświadczający tożsamość podmiotu końcowego. Zadaniem urzędu certyfikacji jest również dostarczanie aktualnych informacji w trybie online, o unieważnieniu danego certyfikatu, by zaufane strony były w stanie zweryfikować jego ważność.

Strona ufająca

Mianem *strony ufającej* określa się odbiorcę certyfikatu. Technicznie rzecz biorąc są to przeglądarki internetowe, systemy operacyjne oraz inne programy, które przeprowadzają walidację certyfikatu. W szerszym ujęciu, strona ufająca jest użytkownikiem końcowym, wykorzystującym certyfikat do bezpiecznej komunikacji w sieci.

Cykl życia certyfikatu rozpoczyna się przygotowaniem, a następnie wysłaniem żądania podpisania certyfikatu (*Certificate Signing Request* (CSR)) do wybranego urzędu certyfikacji. Głównym zadaniem żądania *CSR* jest dostarczenie odpowiedniego klucza publicznego oraz przedstawienie odpowiadającego mu klucza prywatnego. Żądanie zawiera również dodatkowe metadane, aczkolwiek w praktyce, nie wszystkie z nich są używane. Urzędy certyfikacji często nadpisują dane dostarczane w ramach *CSR* informacjami zawartymi w certyfikacie. Otrzymawszy żądanie podpisania certyfikatu, urząd certyfikacji rozpoczyna proces walidacji. W zależności od wybranego typu żądanego certyfikatu, proces ten może może składać się z różnych kroków.

Domain validation

Podstawą wystawienia certyfikatu DV (*Domain validated*) jest przedstawione prawo podmiotu do posługiwania się daną domeną. Potwierdzenie posiadania takiego prawa, odbywa się najczęściej poprzez wysłanie potwierdzającej wiadomości email, na jeden z zatwierdzonych adresów. Jeżeli taka forma potwierdzenia nie jest możliwa, wykorzystane w tym celu może zostać inne medium komunikacji, bądź praktyczna demonstracja posiadanych praw. W przypadku potwierdzenia przez odbiorcę wiadomości posiadanego przez jej nadawcę prawa, wydawany jest certyfikat, który zapewnia jedynie szyfrowanie komunikacji. Certyfikat typu DV nie potwierdza tożsamości jego właściciela.

Organization validation

Certyfikat OV (*Organization validated*) wymaga potwierdzenia tożsamości i autentyczności. Urzędy certyfikacji przed wystawieniem certyfikatu tego typu, ustalają oficjalną nazwę i lokalizację danej organizacji. Następnie, podejmując kolejne kroki w procesie weryfikacji, kontaktują się z daną organizacją celem potwierdzenia złożenia przez nią żądania wystawienia certyfikatu oraz tego, że wnioskodawca jest upoważniony do otrzymania certyfikatu w imieniu danej organizacji. Użytkownik odwiedzający witrynę posiadającą certyfikat OV, może użyć go w celu weryfikacji tożsamości organizacji, zatem podmiotu posiadającego dany certyfikat.[20]

Extended validation

W związku z bardzo restrykcyjnymi wymaganiami, certyfikat EV (*Extended validation*) stanowi najbardziej ceniony typ certyfikatu. Został skonstruowany w taki sposób, by wyeliminować brak spójności certyfikatów OV. Podczas weryfikacji certyfikatu EV, właściciel strony przechodzi przez gruntowny i globalnie znormalizowany proces weryfikacji tożsamości, podczas którego zobligowany jest udowodnić posiadanie wyłącznego prawa do korzystania z domeny, jej legalności, fizycznego istnienia oraz przedstawienia dowodu na wydanie przez podmiot zezwolenia na wydanie certyfikatu.[15]

Po pomyślnym zakończeniu procesu weryfikacji przez urząd certyfikacji, certyfikat zostaje wydany. Oprócz samego certyfikatu, wnioskodawca otrzymuje wszystkie certyfikaty prowadzące do ostatniego ogniwa łańcucha certyfikatu oraz opcjonalnie, instrukcje wdrożenia go na główne platformy. Wnioskodawca, zwany również podmiotem końcowym, po wdrożeniu certyfikatu zapewnia, że komunikacja z jego serwerem w ramach udzielanych przez niego usług jest bezpieczna. W przypadku przechwycenia, bądź złamania odpowiadającego kluczowi publicznemu klucza prywatnego, certyfikat jest unieważniany i podmiot końcowy, w celu zapewnienia bezpieczeństwa udzielanych usług, musi wnioskować o wydanie kolejnego certyfikatu. [1, Rozdz. 3, *Certificate Lifecycle*]

Rozdział 3

Wektory ataków na infrastrukturę klucza publicznego (PKI)

Infrastruktura klucza publicznego narażona jest na wiele zagrożeń, w związku z istnieniem dużej ilości wektorów ataków. W większości przypadków, są one wymierzone w proces walidacji. Potencjalny atakujący może otrzymać certyfikat, będąc w stanie przekonać urząd certyfikacji o posiadaniu praw do danej domeny, których rzeczywiście nie posiada. W przypadkach, kiedy celem ataku są zabezpieczenia samego urzędu, może mieć miejsce sytuacja, w której atakujący złamawszy je, będzie miał możliwość wystawienia fałszywie zaufanego certyfikatu dla dowolnej domeny. Rozdział trzeci stanowi historyczny przegląd wektorów ataków wymierzonych w infrastrukturę klucza publicznego. [1, Rozdz. 4]

3.1 Microsoft i certyfikaty VeriSign

W styczniu 2001 roku, firma VeriSign wystawiła dwa certyfikaty do cyfrowego podpisywania kodu osobie, podającej się za pracownika firmy Microsoft. Osoba o fałszywej tożsamości, musiała w tym celu udowodnić autentyczność żądania nowych certyfikatów oraz uiścić opłatę w wysokości 800$ za dwa certyfikaty. Oszustwo zostało wykryte kilka tygodni później w marcu, podczas rutynowego audytu.

System Windows nie przyznał fałszywym certyfikatom żadnego ze specjalnych poziomów zaufania. Programy podpisane przez certyfikaty cyfrowo, nie uruchomiłyby się jednak bez wyświetlenia ostrzeżenia, w którego treści znajdowałaby się informacja o potwierdzeniu autentyczności programu przez Microsoft, co niewątpliwie stanowiłoby istotny komunikat, który mógł zaważyć o decyzji i nakłonić użytkownika do uruchomienia programu.

Natychmiastowe unieważnienie fałszywych certyfikatów przez firmę Veri-

Sign, nie rozwiązało problemu całkowicie. W związku z brakiem informacji o unieważnieniu w żadnym z dwóch certyfikatów, firma Microsoft była zmuszona wydać awaryjną aktualizację, umieszczającą fałszywe certyfikaty na czarnej liście oraz wyjaśniającą użytkownikom systemu Windows, jak je rozpoznać. Zaistniała sytuacja rozpoczęła ożywione dyskusje dotyczące implementacji mechanizmu unieważniania certyfikatów w systemie Windows. [1, Rozdz. 4, *VeriSign Microsoft Code–Signing Certificate*]

3.2 Thawte login.live.com

Kolejny atak na mechanizm walidacji certyfikatów miał miejsce w marcu 2008 roku. Security researcher Mike Zusman, oszukał ten proces i uzyskał od firmy Thawte certyfikat dla domeny *login.live.com* (będącą domeną Microsoftu), która dla milionów użytkowników stanowi centrum uwierzytelniania za pomocą pojedynczego logowania.

Wektor ataku na proces walidacji certyfikatów w firmie Thawte wykorzystywał fakt iż wystawiała ona certyfikaty typu DV na podstawie uwierzytelnienia za pomocą wiadomości email oraz fakt, że Microsoft zezwala na założenie dowolnym osobom adresu email na skrzynce *@live.com*. Po założeniu przez Zusmana konta z adresem *sslcertificates@live.com*, otrzymanie certyfikatu dla domeny było formalnością.

Zusman ujawnił problem w sierpniu 2008, natomiast nazwę skompromitowanego urzędu certyfikacji wyjawił dopiero rok później. W 2015 roku, firmę Microsoft spotkała ponownie taka sama sytuacja, tym razem dotyczyła ona domeny *live.fi*. [1, Rozdz. 4, *Thawte login.live.com*]

3.3 StartCom

W grudniu 2008 roku, Mike Zusman zdołał obejść proces walidacji nazwy domeny wykorzystując podatność aplikacji internetowej firmy StartCom. Proces walidacji składał się z dwóch etapów. W pierwszym z nich należało udowodnić prawo do kontrolowania i zarządzania domeną, następnie w drugim należało żądać certyfikatu. Odkrycie luki w aplikacji kontrolującej wystawianie certyfikatów, umożliwiło mu uzyskanie certyfikatów dla dwóch domen, do których nie był upoważniony.

Wykonany przez niego atak, został sprawnie wykryty. Szybka reakcja była wynikiem tylko tego, że próbował on uzyskać certyfikaty dla domen *paypal.com* oraz *verisign.com*. Okazało się iż StartCom posiadał wtórny mechanizm kontrolny, który stanowiła czarna lista wysoko profilowych stron internetowych. W momencie wykonania działań przez Zusmana, mechanizm ten spowodował natychmiastowe unieważnienie obydwu fałszywych certyfikatów. [1, Rozdz. 4, *StartCom Breach (2008)*]

3.4 Certyfikat CertStar dla Mozilli

Kilka dni później, po wykonaniu przez Zusmana ataku na StartCom, dyrektor do spraw technicznych (CTO) oraz dyrektor operacyjny (COO) tej firmy, odkryli podobny problem dotyczący kolejnego urzędu certyfikacji. Wiadomości email oznaczone jako SPAM, stanowiące jedyną poszlakę w całym dochodzeniu, informowały o możliwości odnowienia certyfikatu przez firmę CertStar (partnera firmy Comodo), bez konieczności jakiejkolwiek walidacji nazwy domeny. Za pomocą usług firmy CertStar, Eddy Nigg (CTO), uzyskał certyfikaty dla domen *startcom.org* oraz *mozilla.org*. Fałszywy certyfikat przyznany dla wysoko profilowej nazwy domeny Mozilli odbił się głośnym echem w prasie oraz wzbudził żywe dyskusje na forach dyskusyjnych.

Weryfikacja wszystkich 111 certyfikatów wystawionych przez Certstar doprowadziła do unieważnienia przez Comodo 11 certyfikatów, których autentyczności nie udało się potwierdzić oraz wobec których stwierdzono iż nie istniały podejrzenia, by którykolwiek z nich był rzeczywiście fałszywy. [1, Rozdz. 4, *CertStar (Comodo) Mozilla Certificate*]

3.5 RapidSSL i atak typu chosen–prefix

Prowadzona przez Alexa Sotirova i Marca Stevensa grupa badaczy, wykonała w 2008 roku spektakularny, dowodzący słuszności koncepcji atak na internetową infrastrukturę klucza publicznego. W jego wyniku, zdołali uzyskać fałszywy certyfikat identyfikujący urząd certyfikacji, który mógł zostać użyty do podpisania certyfikatu dowolnej strony internetowej na świecie. Po wydaniu książki dotyczącej badań na temat kolizji certyfikatów w 2006 roku, Marc Stevens i inni badacze z jego zespołu, kontynuowali pracę nad udoskonaleniem techniki ataku typu *chosen–prefix*. Prace prowadzone były w obrębie jednego, prywatnego centrum certyfikacji, w zamkniętym, w pełni kontrolowanym środowisku, ze względu na występowanie ograniczeń uniemożliwiających fałszerstwo w środowisku rzeczywistym.

Atak typu chosen–prefix

Celem ataku jest stworzenie dwóch dokumentów z takim samym podpisem cyfrowym wygenerowanym przez algorytm MD5. W związku z faktem iż większość algorytmów kryptograficznych wykorzystywanych do generowania podpisów cyfrowych podpisuje skróty danych, zamiast danych bezpośrednio. Mając dwa dokumenty, których skrót wygenerowany algorytmem MD5 jest taki sam, podpis cyfrowy jednego dokumentu będzie identyczny do podpisu drugiego dokumentu. W takiej sytuacji, w celu otrzymania zaufanego podpisu cyfrowego fałszywego dokumentu, należy wysłać oryginalny, legalny dokument do odpowiedniej instytucji, by po otrzymaniu podpisu skopiować go do fałszywego dokumentu.

Bardziej skomplikowana jest procedura fałszerstwa certyfikatu, w związku z tym iż urzędy certyfikacji nie przyjmują gotowych certyfikatów do podpisu, a generują je, na podstawie przesłanych do nich danych. Wektor ataku wykorzystujący kolizję certyfikatów polega na użyciu dwóch specjalnie skonstruowanych bloków kolizji, manipulujących algorytmem szyfrującym w celu doprowadzenia go do takiego samego stanu, w przypadków dwóch różnych danych wejściowych. W momencie działania algorytmu, bloki kolizji cofają zmiany w dokumentach, co oznacza, że należy wcześniej poznać prefiks oryginalnego, niesfałszowanego dokumentu, by później umieścić w nim jeden z bloków kolizyjnych. Umieszczenie bloków kolizji na samym końcu nie jest możliwe w praktyce, w związku z czym, pliki wynikowe muszą mieć również identyczne sufiksy.

Konstrukcja kolidującego certyfikatu

Wykorzystanie techniki *chosen–prefix* w rzeczywistości, wymaga dostosowania się do warunków w niej panujących. Atak należy zatem przeprowadzić z uwzględnieniem ograniczeń dotyczących dokumentu, który chcemy sfałszować oraz ograniczeń procesu, w którym dokument jest tworzony i podpisywany cyfrowo. Narzuconymi restrykcjami w kontekście podpisów cyfrowych, są:

1. Certyfikaty tworzone są przez urzędy certyfikacji na podstawie danych wysłanych w żądaniu wystawienia certyfikatu (CSR).

2. Ogólna struktura certyfikatu narzucona jest przez specyfikację *X.509 v3*, której fałszerz nie może zmienić, ani naruszyć, a jedynie przewidzieć.

3. Fałszerz kontroluje jedynie niektóre informacje, które zawarte zostaną w certyfikacie. Skopiowaną bezpośrednio z CSR daną jest klucz publiczny. Umożliwia to fałszerzowi przekazanie urzędowi pozornie losowo wyglądającego bloku kolizji, jako klucza publicznego.

4. W niektórych przypadkach fałszerz może być w stanie zmienić pewne informacje dodane później przez urząd certyfikacji (takie jak data upływu ważności certyfikatu).

Z tych informacji wynika iż prefiks kolizji będzie zawierał wszystkie pola certyfikatu pojawiające się przed sfałszowanym kluczem publicznym, w którym przechowywany będzie zawarty w żądaniu CSR blok kolizji. W związku z tym iż zawartość bloku kolizji zależy od prefiksu, przed stworzeniem danych kolizji oraz wysłaniem ich do urzędu certyfikacji, cały prefiks musi być znany. Posiadając wiedzę na temat struktury certyfikatu, fałszerz jest w stanie przewidzieć wartości większości pól certyfikatu, oprócz dwóch, kontrolowanych przez urząd certyfikacji – numer seryjny oraz data upływu ważności certyfikatu.

W tym przypadku, ustalenie przez badaczy wartości obydwu pól było wynikiem szczęśliwego zrządzenia losu oraz lekkiej pomocy ze strony urzędu certyfikacji. Proces wystawiania certyfikatu przez RapidSSL był w pełni zautomatyzowany i zawsze trwał 6 sekund od momentu otrzymania żądania CSR, co umożliwiło przewidzenie wartości jednego z dwóch kontrolowanych przez urząd pól – daty upływu ważności certyfikatu. Kolejną wartość – zatem numer seryjny – udało się ustalić na podstawie informacji o tym, że numer seryjny nadawany z wykorzystaniem inkrementacji. Zażądanie w takim przypadku dwóch certyfikatów, w bardzo krótkim odstępie czasu, pozwoliło przewidzieć numer seryjny drugiego certyfikatu.

Kolejnym etapem procesu jest ustalenie sufiksu. W związku z tym iż składa się on z kilku znanych rozszerzeń X.509, przy odpowiednich operacjach algorytmu MD5 na blokach danych, sufiks jest identyczny w obu certyfikatach.

Wektor ataku jest zatem następujący:

1. Należy ustalić prefiks generowanego przez urząd certyfikatu oraz wartości pól, które powinny znaleźć się w żądaniu CSR.
2. Należy skonstruować żądany prefiks fałszywego certyfikatu.
3. Następnie należy ustalić sufiks.
4. Skonstruować blok kolizji na podstawie danych z poprzednich etapów ataku.
5. Sporządzić i wysłać żądanie CSR do urzędu certyfikacji.
6. Skonstruować fałszywy certyfikat poprzez połączenie sfałszowanego prefiksu, drugiego bloku kolizji, sufiksu oraz podpisu cyfrowego legalnego, niesfałszowanego certyfikatu.

Wygenerowanie pojedynczej kolizji trwało około 24 godzin oraz wymagało wykorzystania klastra składającego się z dwustu konsoli PlayStation 3. W związku z czym, badacze musieli wybrać odpowiedni moment na wysłanie żądania CSR oraz przewidzieć numer seryjny certyfikatu. Sukces został osiągnięty dopiero po czwartej próbie. [1, Rozdz. 4, *RapidSSL Rogue CA Certificate*]

3.6 Naruszenia bezpieczeństwa odsprzedawców Comodo

Seria kolejnych incydentów rozpoczęła się w marcu 2011 roku. Pierwszy z nich miał miejsce 15 marca. W wyniku oszustwa jednego z urzędów rejestracji firmy Comodo, zostało wydanych 9 certyfikatów dla 7 stron internetowych:

addons.mozilla.org, *global trustee*, *google.com*, *login.live.com*, *login.skype.com*, *login.yahoo.com*, *mail.google.com*. Wyłączając certyfikat wystawiony dla *global trustee*, pozostałe dotyczyły witryn kluczowych dla internetu oraz milionów użytkowników odwiedzających je codziennie. Atak został sprawnie wykryty i szczęśliwie zażegnany w ciągu kilku godzin, czego wynikiem było unieważnienie wszystkich certyfikatów. Nie było jednak jasne, czy wszystkie certyfikaty zostały odzyskane przez atakującego, w związku z faktem iż mechanizm OCSP[19] odnotował informacje o unieważnieniu jedynie certyfikatu Yahoo.

Kolejnego dnia, firma Comodo rozpoczęła proces nanoszenia poprawek bezpieczeństwa i kontaktowała się z zainteresowanymi stronami. Oszukany urząd bezpieczeństwa nie został zidentyfikowany, podczas gdy sprawca ataku twierdził iż była to włoska firma Instant SSL. Informacje o ataku zostały ujawnione przez Comodo i inne poszkodowane firmy. Ujrzały one światło dzienne 22 marca. Co ciekawe, kilka dni przed atakiem, garstka ludzi dowiedziała się o nim ze wskazówek zawartych w publicznie dostępnym kodzie źródłowym przeglądarki Chrome.

W kolejnych dniach marca, Comodo odnotowało dwa kolejne oszustwa, z których jedno okazało się być fałszywym alarmem, natomiast drugie dotyczyło faktycznego ataku, w wyniku którego nie został wystawiony jednak żaden fałszywy certyfikat. Powodem udaremnienia oszustwa mogło być wdrożenie poprawek bezpieczeństwa po atakach z 15 marca.

Również w marcu 2011 roku, na stronie internetowej jednego z odsprzedawców tej firmy – ComodoBR – została wykryta podatność SQL Injection. Została ona wykorzystana przez atakującego do pozyskania prywatnych danych klientów (włączając w nie dane żądań CSR). W wyniku ataku, firma nie poniosła innych konsekwencji związanych z PKI.

Seria incydentów wykazała, że zarządzanie tak złożonym i skomplikowanym ekosystemem jak infrastruktura klucza publicznego, na podstawie jedynie zaufania, jest całkowicie niemożliwa. W wyniku tych wydarzeń, zarządzono by żaden z odsprzedawców nie był upoważniony do wystawienia certyfikatu bez uprzedniej walidacji przez firmę Comodo. [1, Rozdz. 4, *Comodo Resellers Breaches*]

3.7 StartCom

Latem 2011 roku ponownie zaatakowana została firma StartCom. Prawdopodobnym sprawcą ataku był ComodoHacker, odpowiedzialny również za ataki wymierzone w firmę Comodo, wcześniej w marcu tego roku. W związku z atakami, firma Comodo była zmuszona zaprzestać wystawiania certyfikatów na około tydzień. Komunikat umieszczony na stronie internetowej StartCom informował o tymczasowym zawieszeniu działalności przez firmę. Zawierał on również istotną informację dotyczącą wszystkich właścicieli certyfikatów

oraz użytkowników odwiedzających witryny posiadające certyfikaty wystawione przez StartCom, mówiącą o tym iż żaden z certyfikatów nie został w jakikolwiek sposób zagrożony.

W wyniku ataku nie został wydany ani jeden fałszywy certyfikat, natomiast działania ComodoHackera, który mógł uzyskać dostęp do danych wrażliwych oraz zbliżyć się do pozyskania cennego klucza głównego tego urzędu certyfikacji, nie spowodowały żadnych znaczących długoterminowych szkód. [1, Rozdz. 4, *StartCom Breach (2011)*]

3.8 DigiNotar

Holenderska firma DigiNotar pełniąca rolę urzędu certyfikacji oraz obsługująca aspekty PKI holenderskiego e–rządowego programu *PKIoverheid*, w 2011 roku stała się pierwszym urzędem certyfikacji, który został w pełni skompromitowany. Prawdopodobnie w wyniku ataków typu Man in The Middle, zostały wykorzystane fałszywie wystawione przez firmę certyfikaty, które unieważnione zostały później, we wrześniu tego roku.

Ujawnienie ataku

Incydent ujrzał światło dzienne 27 sierpnia, gdy irański użytkownik Gmaila zgłosił sporadyczne problemy napotykane podczas prób uzyskania dostępu do swojego konta. Przestoje w świadczeniu usługi trwały nawet do godziny w ciągu dnia. Ich powodem był nietypowy komunikat ostrzegawczy dotyczący certyfikatu strony. Opisana przez użytkownika sytuacja była wynikiem próby wykonania ataku typu Man in the Middle, która dzięki mechanizmowi „przypinania" klucza publicznego[26] przeglądarki Chrome, została wykryta i udaremniona. Jak okazało się w kolejnych dniach, zgłoszony incydent był częścią ataku na niespotykaną jak dotąd skalę, dotyczącego ponad 300 tysięcy adresów, z których znaczna większość pochodziła z Iranu.

Upadek centrum certyfikacji

W obliczu poważnego kryzysu dotyczącego bezpieczeństwa firmy i jej produktów, holenderski rząd zdecydował o natychmiastowym przejęciu kontroli nad firmą DigiNotar, zatrudniając zewnętrznego konsultanta do spraw bezpieczeństwa cybernetycznego, w celu przeprowadzenia śledztwa.

Wstępny raport firmy Fox–IT przedłożony został 5 września. W raporcie udokumentowano takie usterki jak złośliwy, wykrywalny przez antywirusy kod na najistotniejszych serwerach firmy DigiNotar, brak, bądź wadliwe działanie mechanizmu izolującego najważniejsze komponenty systemu. Ponadto, odnotowano przesłanki o ułatwionym dostępie do serwerów urzędu poprzez zarządzającą sieć lokalną, mimo fizycznie bardzo bezpiecznego umieszczenia

ich w zagrożonym środowisku. W związku z faktem iż wszystkie serwery były członkami jednej domeny Windows, możliwym było uzyskanie dostępu do poszczególnego serwera przy użyciu jednej pozyskanej kombinacji loginu oraz hasła. Ustawione na serwerze hasło nie było relatywnie najsilniejsze, w związku z czym, mogło zostać w prosty sposób złamane atakiem *brute–force*. Zainstalowane na publicznych serwerach oprogramowanie było przestarzałe i ze względu braku aktualizacji, podatne było na liczne wektory ataków. Złośliwe oprogramowanie wykryte podczas audytu bezpieczeństwa, nie mogło zostać usunięte, ze względu na brak ochrony antywirusowej na badanych serwerach. W momencie przedłożenia wstępnego raportu, niejasna pozostała kwestia systemu IPS *Intrusion Prevention System*, który nie zareagował na zewnętrzne ataki blokując je.

Pełny raport oficjalnie wydany został w sierpniu 2012 roku (około rok później po przedłożeniu wstępnego raportu). Składający się z około 100 stron dokument, stanowił najdokładniejszy w historii raport dotyczący kompromitacji urzędu certyfikacji. Wynikało z niego iż pierwszy atak miał miejsce 17 czerwca, kiedy zaatakowano publiczny serwer, na którym uruchomiona była podatna aplikacja zarządzającą treścią. Atakującemu zajęło zaledwie kilka dni, by włamać się do najlepiej zabezpieczonego segmentu sieci, w którym umieszczone były główne zasoby. Segment nie był bezpośrednio podłączony do sieci Internet, aczkolwiek atakujący był w stanie dostać się do niego z mniej istotnych systemów.

Pierwsza partia 128 fałszywych certyfikatów została wydana 10 lipca 2011 roku – około tygodnia później od momentu uzyskania dostępu do serwerów urzędu certyfikacji. W wyniku wydania kolejnych partii, wystawionych zostało około 531 certyfikatów dla 53 unikalnych tożsamości. Ze względu na skalę incydentu, nie jest znana faktyczna liczba sfałszowanych certyfikatów, w związku z modyfikacją rejestrów. Duża liczba wykrytych później certyfikatów, nie mogła zostać odnaleziona w odpowiednich bazach danych.

Jak okazało się później, inna firma zajmująca się usługami w zakresie konsultacji spraw związanych z bezpieczeństwem cybernetycznym, wykryła atak 19 lipca, następnie podjęła próbę oczyszczenia systemu, która zakończyła się przed końcem danego miesiąca. Będąc pewnym zażegnania niebezpieczeństwa, w celu uporania się z jego skutkami, DigiNotar unieważniło niewielką liczbę certyfikatów (ze względu na posiadanie wiedzy jedynie o takiej ilości wydanych w wyniku włamania certyfikatów), lekkomyślnie nie informując jednocześnie o tym nikogo. Zważając na skalę kryzysu, wątpliwym jest by natychmiastowe ujawnienie przejęcia kontroli nad urzędem, uratowało go przed upadkiem. Niewątpliwym jednak jest iż taki ruch powstrzymałby atakujących przed wykorzystaniem fałszywych certyfikatów, ze względu na to iż każdy z nich posiadał osadzoną w nim informację, która pozwalałby mechanizmowi OCSP na śledzenie procesu wdrażania certyfikatu.

Początkowo, logi systemowe zawierały informacje o jedynie kilku żąda-

niach, które prawdopodobnie były wynikiem testów wykonanych przez hakera. Informacje o kolejnych żądaniach zaczęły pojawiać się od 4 sierpnia, co stanowiło pierwsze oznaki masowego wdrażania certyfikatów. Liczba żądań stopniowo wzrastała aż do 29 sierpnia, w którym to dniu przeglądarki internetowe unieważniły certyfikację urzędu DigiNotar, unieważniając tym samym wszystkie fałszywe certyfikaty. Informacje uzyskane od poszkodowanych w wyniku ataku użytkowników, twierdzących iż przejawiał on się seriami, pozwoliło na wyciągnięcie wniosków i wysnucie założeń, jakoby taki sposób ataku spowodowany był ograniczeniami wiążącymi się z wykorzystanym wektorem ataku. Za najbardziej prawdopodobny wektor uznano zatruwanie DNS[13]. [1, Rozdz. 4, *DigiNotar*]

3.9 DigiCert Sdn. Bhd.

Malezyjska firma DigiCert Sdn. Bhd. (niepowiązana z firmą DigiCert mieszczącą się na terenie Stanów Zjednoczonych) pełniąca rolę pośredniczącego urzędu certyfikacji, w wyniku ataku w listopadzie 2011 roku, wystawiła 22 certyfikaty uznane jako słabe oraz pozbawione kluczowych aspektów, zatem:

- **Słaby, 512 bitowy klucz** – klucz o danej długości może być w relatywnie prosty sposób zrefaktoryzowany, przy użyciu jedynie metody *brute–force*
- **Brak narzuconych restrykcji dotyczących użytkowania** – informacja, która zawarta powinna być w rozszerzeniu *Extended Key Usage* (EKU) certyfikatu, warunkowała płaszczyznę, w której użyty mógł zostać dany certyfikat. Umownie, DigiCert Sdn. Bhd. posiadało ograniczenie, by wydawane certyfikaty używane były jedynie w płaszczyźnie stron internetowych, jednak w związku z brakiem informacji o danych restrykcjach w niektórych certyfikatach, mogły one zostać użyte na przykład, do podpisania kodu programu.
- **Brak informacji o unieważnieniu** – w związku z faktem iż żaden z 22 certyfikatów nie posiadał tej informacji, po upływie terminu jego ważności, żaden z nich nie mógł zostać wiarygodnie unieważniony.

Jak okazało się później, źródłem problemu był jeden z kluczy publicznych, który został złamany metodą *brute–force*. Został on użyty do podpisania cyfrowo złośliwego oprogramowania[18]. Współpracujące z DigiCert Sdn. Bhd. firmy – Entrust oraz CyberTrust – w ciągu tygodnia unieważniły wszystkie pośredniczące certyfikaty oraz poinformowały wszystkich dostawców przeglądarek, by mogli wydać odpowiednie aktualizacje oraz umieścić słabe i unieważnione pośredniczące certyfikaty na czarnych listach. [1, Rozdz. 4, *DigiCert Sdn. Bhd.*]

3.10 *Flame*

W maju 2012 roku, grupa badaczy bezpieczeństwa rozpoczęła pracę nad analizowaniem nowego pasma malware'u, które ówcześnie zaatakowało głównie państwa Bliskiego Wschodu. *Flame* stanowiło wtedy najbardziej zaawansowane złośliwe oprogramowanie tego typu. Jego rozmiar wynosił ponad 20 MB i zawierał ponad 20 modułów umożliwiających różne wektory ataku, takich jak między innymi: moduł do przechwytywania transmisji danych w sieci, aktywacji mikrofonu czy odzyskiwania plików. Oprogramowanie wykorzystywało relacyjną bazę danych *SQLite* oraz skryptowy język programowania *Lua*. Zostało napisane w sposób, by pozostać niewykrywalnym przez długi okres.

Irański CERT[7], wydał w maju 2012 roku oficjalne oświadczenie dotyczące malware'u *Flame*. Do tego czasu odnotowano infekcję około 1000 systemów. Niedługo później, twórcy oprogramowania wywołali komendę, która wymusiła samoistne usunięcie każdej uruchomionej instancji malware'u. Mimo tego, odnotowywane były kolejne jednostki złośliwego oprogramowania.

Flame i Windows Update

Malware był w stanie rozpowszechnić się na dowolny komputer z systemem Windows w lokalnej podsieci, wykorzystując podatność mechanizmu Windows Update. W związku z obsługiwaniem przez przeglądarkę Internet Explorer funkcjonalności *Web Proxy Autodiscovery* (WPAD) – protokołu, który może być wykorzystywany przez programy w celu odnalezienia serwerów proxy HTTP w sieci lokalnej – atakujący mógł uruchomić serwer podający się jako proxy, w celu uzyskania dostępu do szyfrowanej komunikacji ofiary. Malware wykorzystywał dokładnie ten wektor ataku, by rozpowszechnić wiadomość o nowych aktualizacjach systemu Windows zawierających złośliwy kod. W związku z uznaniem zabezpieczenia mechanizmu aktualizacji, zatem podpisania cyfrowo przez Microsoft plików binarnych aktualizacji, za wystarczające, dana podatność nie powinna mieć racji bytu. Twórcy malware'u, zdołali jednak sfałszować podpis cyfrowy Microsoftu, co umożliwiło im wykorzystanie tego wektoru ataku.

Flame i Windows Terminal Services

Złośliwe oprogramowanie wykorzystało również luki w usłudze Windows Terminal Services. W celu licencjonowania usługi, każdej instalacji *Terminal Server*, przed aktywacją przyznawano podrzędne certyfikaty, które wykorzystywane były do tworzenia licencji dla użytkowników końcowych. Firma Microsoft popełniła jednak kilka poważnych błędów, projektując ten mechanizm:

1. Główny certyfikat *Terminal Services* wykorzystywany do wystawiania podrzędnych certyfikatów indywidualnym klientom, wydany został na

podstawie tego samego źródłowego certyfikatu (*root certificate*), co dotyczący aktualizacji Windows Update.

2. Oprócz wykorzystania nadrzędnego certyfikatu *Terminal Services* do licencjonowania, mógł on zostać z niewyjaśnionych przyczyn użyty również, do cyfrowego podpisywania kodu.

3. Wydane podrzędne certyfikaty nie miały narzuconych restrykcji dotyczących płaszczyzny wykorzystania, w związku z czym, odziedziczały ograniczenia od nadrzędnego certyfikatu.

Na tej podstawie, każdy użytkownik usługi *Terminal Services* posiadający podrzędny certyfikat, który nie posiadał narzuconych restrykcji, miał możliwość podpisu cyfrowego plików binarnych aktualizacji Windows Update. Podatność mogła zostać wykorzystana jedynie w stosunku do systemów Windows XP, w związku z ignorowaniem przez nie rozszerzenia certyfikatu – *Hydra*. Kolejne wersje systemu Windows nie posiadały danej podatności.

Flame i algorytm MD5

Kolejny poważny błąd popełniony przez firmę Microsoft dotyczył licencjonowania *Terminal Servers*. Używane w tym celu były podpisy cyfrowe generowane przy użyciu algorytmu MD5, który w momencie projektowania mechanizmu, uznawany był za relatywnie niepewny. *Flame* wykorzystywał zatem wektor ataku znany z incydentu związanego z urzędem certyfikacji RapidSSL – *chosen–prefix*. Jednakże w związku z faktem iż w tym przypadku, numer seryjny nie był tak łatwy do przewidzenia jak w przypadku RapidSSL, atakujący musieli posiadać bardzo dobre łącze, w celu zminimalizowania jittera[17]. [1, Rozdz. 4, *Flame*]

3.11 POODLE

POODLE (*Padding Oracle On Downgraded Legacy Encryption*), jest wektorem ataku wykorzystującym lukę protokołu SSL 3.0, wynikającą ze sposobu w jaki szyfrowane są bloki danych z wykorzystaniem typu algorytmów szyfrowania CBC, przez daną wersję protokołu SSL. Atak wykorzystuje możliwość negocjacji wersji protokołu, by wymusić wybór podatnej wersji, a zatem SSL 3.0, w celu deszyfrowania dowolnej treści wymienianej w ramach danej sesji SSL. Podatność procesu odkodowywania szyfrów wygenerowanych dzięki algorytmom wykorzystującym tryb wiązania CBC polega ułomności projektu tych algorytmów. Luka w mechanizmie umożliwia zmianę wartości paddingu znajdującego się na końcu każdego bloku danych, w związku z czym, przy każdym kolejnym wykonywanym przez algorytm przejściu, powoduje on zmniejszenie bezpieczeństwa szyfru. Deszyfrowanie odbywa się bajt po bajcie oraz generuje

dużą ilość połączeń pomiędzy klientem, a serwerem. Atakujący chcąc wykorzystać ten wektor ataku, musi być w stanie aktywnie podsłuchiwać komunikację pomiędzy stronami.

3.12 TURKTRUST

W grudniu 2012 roku, dzięki mechanizmowi „przypinania" klucza publicznego (*public key pinning*), Google odkryło kolejny poważny problem związany z PKI. Mechanizm umożliwia aplikacji klienckiej – w tym przypadku przeglądarce internetowej – sprawdzenie, czy jedynie autoryzowane urzędy certyfikacji, wydają certyfikaty określonym stronom internetowym. Przeglądarka Chrome posiada wbudowaną, krótką listę najbardziej znanych stron internetowych. W grudniu 2012 roku, użytkownik danej przeglądarki napotkał problem, który spowodowany był tym iż napotkany certyfikat nie pasował do żadnej z pozycji wbudowanej w przeglądarkę listy. W wyniku incydentu, Chrome wysłał informacje o łańcuchu certyfikatu do Google, na podstawie którego pozyskano informacje o urzędzie certyfikatu, który wydał fałszywy certyfikat. Nieprawidłowe podrzędne certyfikaty zostały natychmiastowo unieważnione przez wszystkie strony. Turecki urząd certyfikacji TURKTRUST wydał oficjalne oświadczenie, w którym poinformował o pomyłce, która miała miejsce w sierpniu 2011 roku. W trakcie przechodzenia pomiędzy dwiema instalacjami systemowymi, w wyniku błędu zostały wydane dwa certyfikaty oznaczone jako certyfikaty identyfikujące urząd certyfikacyjny. Incydent nie został wykryty przez około 15 miesięcy, podczas, gdy w tym czasie, zostały one użyte jako certyfikaty serwerów.

Jedna z firm posiadająca nieprawidłowy, podrzędny certyfikat, zainstalowała firewalla z funkcjami MitM. Po zaimportowaniu do zapory certyfikatu, rozpoczęła ona generowanie fałszywych certyfikatów stron internetowych na żądanie. W trakcie trwania procesu, stworzony został klon certyfikatu Google'a, który został wykryty przez przeglądarkę Chrome. Obsługujący głównego zbioru urzędów certyfikujących (*root stores*) przeglądarki, przyjęli tłumaczenie urzędu TURKTRUST, mówiące o tym, że powodem incydentu był błąd administracyjny. Nie wykryto żadnych przesłanek, jakoby incydent był wynikiem ataku na urząd certyfikacji. Mozilla wymusiła na TURKTRUST przeprowadzenie audytu, natomiast Google oraz Opera przestały uznawać certyfikaty wydawane przez ten urząd certyfikaty typu EV. [1, Rozdz. 4, *TURKTRUST*]

3.13 Powszechne przechwytywanie SSL

Mimo wielu słabości infrastruktury publicznego klucza, okazało się iż największym zagrożeniem dla ekosystemu jest powszechne przechwytywanie SSL przez pracowników, lokalnie zainstalowane oprogramowanie lub dostawców interne-

tu. Z biegiem czasu, zjawisko stało się dosyć powszechne, mimo tego iż rzadko kiedy o nim słyszano. W związku z nieświadomością osób, o mających miejsce incydentach, nie są one zgłaszane ani raportowane, przez co świadomość problemu wzrasta w bardzo wolnym tempie.

Superfish

Adrienne Porter Felt była autorką kolejnego znaczącego odkrycia. W lutym 2015 roku, ujawniona została przez nią informacja o dostarczaniu do niektórych z systemów Lenovo, preinstalowanego oprogramowania typu *ad–injector*, o nazwie Superfish. Oprogramowanie przechwytywało cały ruch sieciowy wygenerowany przez użytkownika, włącznie z tym, przeznaczonym dla zabezpieczonych stron internetowych. Superfish, w celu uniknięcia ostrzeżeń dotyczących certyfikatów, użył niechcianego „zaufanego" głównego certyfikatu, dodanego do *root store* systemu operacyjnego bez zgody użytkownika. Ruch przechodzący przez przeglądarkę, przekierowywany był przez oprogramowanie do lokalnego procesu proxy, który pobierał z Internetu żądane treści, wprowadzając do nich dowolne modyfikacje. W związku z takim sposobem działania, Superfish był w stanie obserwować całą komunikację i mieć wgląd we wszystkie dane wymieniane w jej ramach, co budziło wątpliwości i pytania dotyczące sfery moralności. Oprogramowanie obarczone problemem obyczajowym i etycznym, niepozbawione było również problemów stricte technicznych. Superfish wykorzystywał ten sam *root certificate* dla każdego systemu, co było błędną praktyką, w związku z faktem iż taka sytuacja mogła zostać wykorzystana przez potencjalnego atakującego do wyodrębnienia klucza prywatnego i wykorzystania go w celu zaatakowania innych użytkowników posiadających system Lenovo z tą podatnością. Po publicznym ujawnieniu incydentu przez Adrienne Porter Felt, podatność została natychmiastowo usunięta. Niestety nie stanowiła ona jedynej słabości oprogramowania. Wspomniane wcześniej lokalne oprogramowanie proxy uruchomione na każdym laptopie, miało słabsze możliwości dotyczące protokołu TLS niż ówczesne wersje przeglądarek, wspierając protokół TLS 1.1 zamiast najnowszej wersji TLS 1.2. W związku z tym, oprogramowanie proxy obsługiwało dużą ilość słabych zestawów algorytmów szyfrowania, obniżając jednocześnie poziom bezpieczeństwa użytkowników. Co więcej, użytkownicy Superfish, odwiedzając jakąkolwiek stronę internetową, nie otrzymywali ewentualnych ostrzeżeń dotyczących certyfikatów.

Początkowo, firma Lenovo próbowała bronić swoich racji, by ostatecznie zaakceptować nieuniknione. Analizy przeprowadzone przez Facebooka wskazały na to iż działania prowadzone przez oprogramowanie Superfish, dotknęło użytkowników na całym świecie. Szacunkowo, 4.5% połączeń z Facebookiem na terenie Kazachstanu, wykonywanych było przez laptopy Lenovo wyposażone w oprogramowanie Superfish. [1, Rozdz. 4, *Widespread SSL Interception*]

Rozdział 4

Bezpieczeństwo protokołu HTTP

Protokół TLS zaprojektowany został w celu zapewnienia bezpieczeństwa połączeń TCP. Mimo spełnienia tego aspektu, użytkownik narażony jest na liczne wektory ataków, w związku z licznymi problemami wynikającymi z organicznego rozwoju sieci oraz chaotycznymi sposobami interakcji pomiędzy różnymi elementami ekosystemu internetowego. Rozdział czwarty skupia się zatem na relacji pomiędzy protokołami HTTP oraz TLS. [1, Rozdz. 5]

4.1 Sidejacking

W momencie nawiązania komunikacji z serwerem, tworzona jest sesja, którą identyfikuje się na podstawie przypisanego do niej identyfikatora sesji. Osoba postronna, mająca dostęp do komunikacji użytkownika z serwerem, jest w stanie ją przechwycić. Szczególnym przypadkiem przechwytywania sesji aplikacji internetowej, w której identyfikatory sesji pozyskiwane są z nieszyfrowanego strumienia danych jest *sidejacking*. W przypadku, gdy strona internetowa nie zapewnia bezpiecznej komunikacji, w celu wykorzystania tego wektoru ataku, atakujący musi jedynie obserwować niezaszyfrowaną komunikację i wydobyć z niej identyfikator sesji. Natomiast, w przypadku gdy komunikacja jest szyfrowana jedynie częściowo, możliwym jest wystąpienie dwóch rodzajów błędów:

Wyciek sesji – z powodu struktury projektu

Niektóre witryny internetowe używają szyfrowania komunikacji jedynie w celu ochrony haseł, by w momencie uwierzytelnienia użytkownika, przejść ponownie na nieszyfrowaną komunikację. Podejście, które zapewnia nieznaczną poprawę stopnia bezpieczeństwa, efektywnie zapobiegając wyciekom haseł, ciągle umożliwia trywialne przechwycenie identyfikatora sesji. W porównaniu do haseł, tokeny sesji są relatywnie mniej wartościowe, w związku z faktem iż są one ważne jedynie przez określony okres.

Wyciek sesji – z powodu błędu

Usilne próby zapewnienia bezpiecznej komunikacji w obrębie całej strony internetowej, mogą nie przynieść pożądanych rezultatów, w związku z potencjalnie popełnionymi błędami w wykonywanym procesie. W ich wyniku, pozostawione w niezaszyfrowanej formie zasoby danej domeny, odpowiednio wykorzystane mogą powodować wyciek danych dotyczących nawiązanej sesji.

Sidejacking może być wykorzystany w przypadku któregokolwiek sposobu transportowania identyfikatora sesji, w związku z tym iż atakujący posiada pełen dostęp do komunikacji pomiędzy stronami. Wektor ten, może być zatem wykorzystany nie tylko w przypadku, gdy identyfikator sesji zachowywany jest w ciasteczkach, ale również w przypadku umieszczenia go w adresie URL. Pozyskany identyfikator posłuży atakującemu do podszycia się pod użytkownika końcowego, w celu kontynuowania komunikacji z serwerem. [1, Rozdz. 5, *Sidejacking*]

4.2 Kradzież plików cookies

Zapewnienie zaszyfrowanej w 100% komunikacji, warunkuje ukrycie identyfikatora sesji za warstwą szyfrowania, co uniemożliwia wykorzystanie wektoru *Sidejacking*. Jednakże częsty błąd popełniany przez programistów, polegający na niezaszyfrowaniu ciasteczek, otwiera furtkę, która może zostać wykorzystana w celu wykorzystania techniki zwanej *kradzieżą ciasteczek* (cookie stealing).

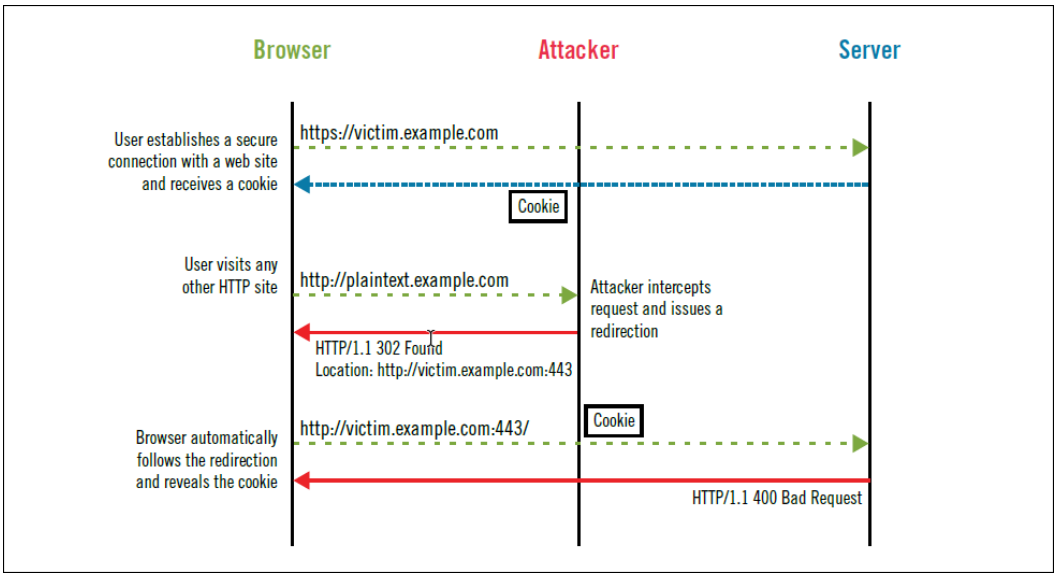
Domyślnie, ciasteczka wykorzystywane są zarówno na porcie 80 jak i 443. Od administratora danej strony internetowej, który zapewnia bezpieczną komunikację z nią, oczekuje się, by pliki cookies również zostały zabezpieczone, by przeglądarki były w stanie odpowiednio je obsłużyć. Natomiast w przypadku, gdy atakujący wymusi na przeglądarce wysłanie żądania w jawnej postaci na port 80, a ciasteczka nie zostały uprzednio oznaczone jako bezpieczne, zostaną one ujawnione.

W istocie, atak jest stosunkowo prosty. Atakujący aktywnie obserwując komunikację ofiary, oczekuje na wywołanie przez nią nieszyfrowanego żądania do dowolnej strony internetowej. W momencie jego wykonania, jest on w stanie przejąć niezabezpieczone połączenie i przekierować przeglądarkę na port 80 danej strony internetowej, w ramach odpowiedzi na żądanie ofiary. W związku z faktem iż dowolna strona internetowa może wymusić przekierowanie na inną witrynę, przeglądarka zezwala na takie działania atakującego. Ten wektor ataku, możliwy jest nawet w przypadku, gdy serwer HTTP nie jest uruchomiony na porcie 80, w związku z faktem iż atakujący jest w stanie podszyć się pod serwer umożliwiający jawną, nieszyfrowaną komunikację na dowolnym porcie.

Wynikiem działań jest nawiązanie przez przeglądarkę nieszyfrowanego połączenia z docelową stroną internetową oraz przechowywanie przez nią nieza-

bezpieczonych ciasteczek. W przeciwieństwie do typowej przeglądarki, która nie oznacza plików cookies jako bezpieczne, atakujący posiada wtedy wszystkie identyfikatory sesji ofiary i jest w stanie przechwycić sesję.

W celu pozyskania innym sposobem wszystkich informacji dotyczących ciasteczek przechowywanych przez przeglądarkę ofiary, atakujący musi wymusić przekierowanie przeglądarki na tę samą nazwę hosta oraz na port 443. Sytuację tę przedstawia rysunek 4.1. Wymuszenie jawnej, nieszyfrowanej komunikacji, odbyć się powinno zatem przy przekierowaniu ofiary na adres *http://www.example.com:443*. Przypadek, w którym próba nawiązania nieszyfrowanej komunikacji na porcie, który wymaga jej szyfrowania, zakończy się niepowodzeniem, a przeglądarce zwrócony zostanie kod błędu 400 (Bad Request), ciągle jest przypadkiem, w którym atakujący osiąga swój cel. Dzieje się tak, w związku z faktem iż wykonane przez przeglądarkę w taki sposób żądanie, zawiera już niezabezpieczone pliki cookies, a zatem wszystkie pożądane przez atakującego informacje.



Rysunek 4.1: Schemat przedstawiający wektor ataku kradzieży ciasteczek z wykorzystaniem ataku typu MitM [1]

[1, Rozdz. 5, *Cookie Stealing*]

4.3 Manipulacja plikami cookies

Wektor ataku wykorzystywany w przypadku, kiedy atakujący nie jest w stanie uzyskać dostępu do ciasteczek, ze względu na prawidłowe ich zabezpieczenie. Polega na wykorzystaniu słabych punktów specyfikacji plików cookies, które umożliwiają wstrzyknięcie nowych ciasteczek, w wyniku czego stare zostają

nadpisywane, a aktualnie istniejące usuwane. Zapewnienie całkowitego szyfrowania komunikacji, nie zawsze jest zatem gwarantem bezpieczeństwa plików cookies.

Mechanizm ciasteczek został zaprojektowany w celu umożliwienia przechowywania niewielkich ilości danych po stronie klienta. Każdemu nowemu plikowi cookie, przypisywana jest przez serwer para: nazwa–wartość oraz metadane wskazujące na cel oraz czas ważności ciasteczka. W celu stworzenia nowego pliku cookie wykorzystuje się nagłówek odpowiedzi *Set–Cookie*. W przypadku każdej transakcji HTTP, przeglądarki przeszukują tak zwane magazyny ciasteczek (*cookie jars*), w celu odnalezienia odpowiadających danej transakcji plików cookie, by w rezultacie, przesłać je dzięki nagłówkowi odpowiedzi *Cookie*.

Przez długi czas, uboga specyfikacja ciasteczek powodowała istnienie wielu luk oraz niespójności, które mogły zostać wykorzystane w celu wykonania skutecznego ataku. W 2011 roku, wydany został bieżący standard RFC 6265[25].

Z punktu widzenia bezpieczeństwa, występują dwa problemy dotyczące ciasteczek:

1. Ich pierwotny projekt zezwalał i zachęcał do wykorzystania luk źle zaprojektowanego mechanizmu.

2. Pliki cookies nie są zsynchronizowane z głównym mechanizmem bezpieczeństwa, wykorzystywanym przez przeglądarki internetowe – *same–origin policy* (SOP).

Luźne ustalanie zakresu nazw hostów

Pliki cookies przeznaczone są do bycia udostępnianymi pomiędzy wszystkimi nazwami hostów danej domeny, jak i pomiędzy protokołami i portami. Ciasteczko przeznaczone dla domeny *example.com*, będzie widoczne również dla każdej jej subdomeny – np. *secure.example.com*. W przypadku, w którym przykładowa nazwa hosta *blog.example.com* wydaje domyślnie (bez określenia parametru *Domain*) plik cookie jedynie dla siebie, może rozszerzyć jego widoczność dla nadrzędnej domeny *example.com*. W rezultacie, fałszywy serwer jest w stanie wstrzykiwać ciasteczka do innych stron oraz aplikacji zainstalowanych na nazwach hostów, które współdzielą tę samą nazwę domeny. Można je również określić mianem powiązanych nazw hostów bądź stronami pokrewnymi.

Tak luźne podejście w wyznaczaniu zakresu nazw hostów sprzeczne jest z regułami SOP, które definiują ogólny kontekst bezpieczeństwa z dokładnym dopasowaniem protokołu, nazwy hosta oraz portu. Zatem wdrożenie bezpiecznej witryny jest znacznie trudniejsze, w związku z faktem iż pliki cookies mogą zostać przypisane z dowolnej powiązanej nazwy hosta, co znacznie zwiększa obszar ataku.

Serwery nie wykrywają metadanych

W związku z faktem iż serwerom przekazuje się jedynie parę nazwa–wartość ciasteczka, nie znają one jego źródła. W przypadku, w którym serwerowi przekazane zostały odpowiednie metadane, mógłby on odrzucić ciasteczka, które nie zostały przez ten serwer stworzone.

Brak integralności bezpieczeństwa plików cookies

Głównym problemem jest fakt iż pliki cookies pracują płynnie zarówno w protokołach HTTP, jak i HTTPS. Mimo faktu iż oznaczone atrybutem *secure* ciasteczko może zostać przesłane jedynie za pośrednictwem zaszyfrowanego kanału komunikacji, zarówno ciasteczka oznaczone jak i nieoznaczone tym atrybutem, przechowywane są w tym samym obszarze nazw. Co gorsza, dana flaga bezpieczeństwa nie stanowi integralnej części tożsamości ciasteczka. W przypadku, gdy nazwa pliku cookie, domena oraz ścieżka pasują do siebie, oznaczone atrybutem *secure* ciasteczko, zostanie zastąpione niezabezpieczonym plikiem.

Ataki wykorzystujące manipulację plikami cookies

Wyróżnia się trzy typy ataków wykorzystujących manipulację ciasteczkami. Dwa z nich należą do podkategorii *cookie injection*, ze względu na fakt iż w ich rezultacie tworzone są nowe pliki cookies. Trzeci atak zezwala na usunięcie ciasteczek.

Eksmisja plików cookies

Typ ataku wymierzony w magazyny plików cookie, które wymuszają maksymalny rozmiar ciasteczek, ilość dla danej domeny oraz ich łączny rozmiar. Wektor ataku polega na przesłaniu dużej ilości fałszywych ciasteczek, w czego rezultacie, przeglądarka usuwa wszystkie prawdziwe, pozostawiając w magazynie jedynie te wymuszone.

Magazyny plików cookie posiadają liczne restrykcje. Całkowita liczba ciasteczek oraz rozmiar magazynu jest ograniczony, w związku z czym, nakładany jest limit dla każdego hosta, by zapobiec przejęciu całego magazynu przez tylko jednego z nich. Mając na uwadze ograniczenia dotyczące rozmiaru pojedynczego ciasteczka do około 4KB, skuteczne wykonanie tego ataku może wymagać wykorzystania wielu nazw domen, w celu przepełnienia magazynu.

Bezpośrednie wstrzyknięcie pliku cookie

Wektor ataku wykorzystujący fakt iż zarówno zabezpieczone jak i niezabezpieczone pliki cookie, działają w obrębie tej samej przestrzeni nazw. Bezpośrednie wstrzyknięcie plików cookies odbywa się w zaszyfrowanym środowisku, zatem

atakujący nie jest w stanie pozyskać ciasteczek w jawnej, niezaszyfrowanej postaci. Może on jednak istniejące już pliki cookies nadpisać nowymi.

Wektor tego ataku może w teorii przypominać algorytm wykonania ataku kradzieży plików cookie, gdzie atakujący przechwytywał jawną, nieszyfrowaną transakcję HTTP zainicjowaną przez ofiarę, by wymusić jawne żądanie HTTP do danej, docelowej witryny internetowej. Następnie, przechwytywał to żądanie i odpowiadał na nie odpowiedzią HTTP zawierającą dowolne pliki cookies.

W praktyce jednak, w celu skutecznego nadpisania ciasteczka, jego nazwa, domena oraz ścieżka musi zgadzać się z oryginałem. By zdobyć pożądane informacje, atakujący musi obserwować, które wartości metadanych wykorzystywane są przez docelową witrynę internetową, w celu użycia ich w ataku.

Wstrzykiwanie plików cookies z powiązanych nazw hostów

W przypadku, w którym bezpośrednie wstrzyknięcie plików cookies nie jest możliwe, atakujący może wykorzystać fakt iż są one współdzielone przez powiązane nazwy hostów.

Właściciel strony internetowej *www.example.com* oraz *blog.example.com*, bloga hostowanego przez firmę zewnętrzną skupiającą się w relatywnie niewielkim stopniu na bezpieczeństwie, w przypadku odnalezienia przez atakującego podatności XSS (*Cross–site scripting*), jest on narażony na atak tego typu. Atakujący, wykorzystując tę podatność w aplikacji bloga, jest w stanie manipulować plikami cookies głównej aplikacji. Wektor ataku polega na wymuszeniu wysłania żądania HTTP do podatnej strony, na której ciasteczka mogą zostać nadpisane. W przypadku, w którym użytkownik nie posiada jeszcze żadnych plików cookies z docelowej strony, w wyniku ataku, jego przeglądarka wykorzysta wstrzyknięte, fałszywe ciasteczka. Zakładając, że wykorzystany zostanie atak XSS, atakujący musi jedynie wykonać na stronie aplikacji bloga komendę:

```
document.cookie='JSESSIONID=FORCED_ID; domain=example.com';
```

gdzie:

- **FORCED_ID** – jest wstrzykniętym identyfikatorem pliku cookie
- **domain** – jest atrybutem, dzięki któremu atakujący jest w stanie rozszerzyć widoczność pliku cookie aż do *www.example.com*, zatem strony głównej aplikacji

Pozyskiwanie pierwszego ciasteczka

Znacznie bardziej prawdopodobnym jest przypadek, w którym potencjalna ofiara posiada już oryginalne pliki cookies nadane przez stronę. W przypadku

wstrzyknięcia przez atakującego fałszywych ciasteczek, przeglądarka domyślnie zaakceptuje oba pliki, by następnie każde żądanie do docelowej strony, zawierało zarówno oryginalne jak i wstrzyknięte ciasteczko. Dzieje się tak w związku z faktem iż przeglądarka widzi oba pliki, jako dwa niezależne ciasteczka. Mimo tego, atak nie może być kontynuowany, w związku z tym iż w przypadku wielu plików cookies z taką samą nazwą, aplikacja webowa zazwyczaj wybierze i wykorzysta ten plik, który jest dla niej „znany". Atakujący ma w takim przypadku dwa wyjścia. Pierwszym z nich jest podjęcie trudnej próby eksmisji plików cookies z magazynu. Natomiast drugim, alternatywnym wyjściem jest modyfikacja metadanych ciasteczka, by wymusić pierwszeństwo nad oryginalnym plikiem. Taki rezultat można osiągnąć wykorzystując fakt iż przeglądarki wysyłają z pierwszeństwem bardziej szczegółowe pliki cookies:

document.cookie = 'JSESSIONID=SECOND_FORCED_ID;
domain=example.com; path=/admin';

Z danym atrybutem *path* wskazującym na ścieżkę */admin*, przeglądarka wyśle pliki cookies w następującej kolejności:

- Cookie: JSESSIONID=SECOND_FORCED_ID; JSESSIONID=REAL_ID; JSESSIONID=FORCED_ID

Zatem jako pierwsze, zostanie wysłane drugie sfałszowane ciasteczko *SECOND_FORCED_ID*, następnie drugi, oryginalny plik *REAL_ID* oraz na koniec *FORCED_ID*. W przypadku, gdy atakujący chce „zainfekować" wiele sekcji, musi wstrzyknąć adekwatnie dużo ciasteczek dla każdej ścieżki

Nadpisywanie plików cookies z wykorzystaniem powiązanych nazw hostów

Kolejny przypadek, który umożliwia nadpisanie oryginalnych ciasteczek. Wykorzystuje fakt iż docelowe strony internetowe ustawiają ciasteczkom swoją główną domyślną domenę. Dana sytuacja ma miejsce relatywnie rzadko, w związku z tym iż większość stron nie wskazuje explicite domyślnej domeny. Kolejnym powodem niepowodzenia nadpisania plików cookies z poziomu powiązanej nazwy domeny jest fakt iż nie można wydać pliku cookie dla siostrzanej nazwy hosta. Możliwym jest zatem wydanie ciasteczka dla strony *www.blog.example.com* oraz *example.com* ze strony *blog.example.com*, ale niemożliwym jest wydanie z tej strony pliku cookie dla *www.example.com*.

Nadpisywanie plików cookies z wykorzystaniem fałszywie powiązanych nazw hostów

Istnieje jeszcze jeden przypadek, w których atakujący jest w stanie nadpisać oryginalną wartość pliku cookie – strona internetowa explicite wskazuje domenę ciasteczek, która nie musi być domeną główną (*root*). Atakujący metodą Man in the Middle może wybrać powiązane nazwy hostów, które będą celem ataku. W związku z faktem iż rdzeń Internetu działa na nieuwierzytelnionym systemie DNS, atakujący jest w stanie przejąć kontrolę nad tym mechanizmem, by następnie podszyć się pod dowolną domenę. W takim przypadku, chcąc zaatakować stronę *www.example.com*, może stworzyć subdomenę *www.www.example.com*, by następnie stworzyć plik cookie dla strony *www.example.com*.

Wpływ

Wiele stron internetowych zaprojektowanych jest z założeniem iż atakujący nie będzie w stanie wykryć oraz zidentyfikować zawartości plików cookies. Oczywiście dane założenie jest niepoprawne, natomiast przyjęty w celu ataku wektor, zależny jest od konkretnej aplikacji.

XSS

W przypadku, gdy developer nie zakłada zmiany zawartości ciasteczek, mogą być one wykorzystywane w niezabezpieczony sposób, na przykład, mogą zostać użyte do bezpośredniego umieszczenia ich w kodzie HTML, co może prowadzić do stworzenia podatności XSS.

Obejście mechanizmu obronnego CSRF

Niektóre strony internetowe opierają się na mechanizmie ochronnym CSRF[9], który wymaga zgodności tokenu umieszczonego w parametrach strony z tokenem umieszczonym w ciasteczku. Atakujący będąc w stanie wymusić wartość konkretnego pliku cookie, może sprawić, że mechanizm ten stanie się w danym przypadku bezużyteczny.

Zmiana stanu aplikacji

Programiści dosyć często traktują ciasteczka jako bezpieczny od modyfikacji sposób przechowywania niewielkiej ilości danych. W przypadku, gdy pewien etap procesu podejmowania decyzji w aplikacji polega na wartości ciasteczka, w wyniku wykorzystania podatności plików cookies, może ona zostać zmanipulowana.

Fiksacja sesji

Przykład odwrotnego do przechwytywania sesji ataku, w którym zamiast

pozyskania identyfikatora sesji ofiary, atakujący łączy się z docelową stroną internetową uzyskują własny identyfikator, by następnie nakłonić ofiarę do przyjęcia jego *SessionID*.

Środki łagodzące

Ataki z kategorii manipulowania plikami cookies, mogą być udaremniane z wykorzystaniem pewnych środków łagodzących, mających na celu zapobiegnięcia fałszowania ciasteczek oraz zweryfikowania ich oryginalności.

Wdrożenie standardu HTTP Strict Transport Security

HSTS[24] będący relatywnie nowym standardem, wymusza szyfrowanie na nazwie domeny, na której dany standard jest wdrożony oraz opcjonalnie na wszystkich jej subdomenach. Takie rozwiązanie uniemożliwia wykorzystanie ataku typu Man in the Middle i oszustwa systemu DNS, w celu wstrzyknięcia ciasteczek, bez złamania szyfrowania.

Walidacja integralności plików cookies

Walidacja integralności plików cookies stanowi najlepszą linię obrony przeciwko atakom wstrzykiwania ciasteczek. Sposób ten polega na zapewnieniu iż otrzymany od klienta plik cookie pochodzi z konkretnej strony internetowej. W tym celu wykorzystywany może zostać na przykład algorytm *HMAC*.

Bardzo ważnym jest, by schemat walidacji integralności plików cookies został zaprojektowany w taki sposób by ciasteczko wydane konkretnemu użytkownikowi nie było ważne dla innego. W przeciwnym przypadku, atakujący pozyskawszy ciasteczko dla własnego konta, mógłby je wstrzyknąć do konta potencjalnej ofiary.

Walidacja integralności ciasteczek oraz schematy szyfrowania nie są w stanie pomóc w zabezpieczeniu sesyjnych plików cookies, które są skutecznie ograniczony czasowo mechanizmem, wykorzystywanym do zastępowania haseł. Identyfikator kanału (*ChannelID*) stanowi próbę rozwiązania tego problemu, tworząc kryptograficzne wiązanie pomiędzy przeglądarką, a stroną internetową na poziomie TLS. Podejście znane jest również jako *channel binding*. W wyniku jego adaptacji, tworzona jest nowa sesja, która może zastąpić sesję HTTP. W praktyce, bardziej prawdopodobnym jest, że zachowany zostałby istniejący mechanizm oparty o ciasteczka, ale przywiązane do bezpiecznego kanału jako mechanizm ochronny przeciwko przechwytywaniu sesji. [1, Rozdz. 5, *Cookie Manipulation*]

4.4 SSL Stripping

Wektor ataku wykorzystujący fakt nieokreślenia przez użytkownika prefiksu *https://* bądź rozpoczęcia przez niego sesji przeglądania w jawnej, nieszyfrowanej części strony. W związku z tym iż jawna część komunikacji jest w pełni widoczna, może zostać dowolnie zmodyfikowana przez atakującego. Przykładowo, jeżeli strona internetowa zawiera adres do bezpiecznego serwera, atakujący jest w stanie przepisać jej zawartość, w celu zamiany bezpiecznego linku na adres jawnej strony. W związku z brakiem linku do bezpiecznej strony, potencjalna ofiara może nigdy nie trafić zabezpieczonej strefy. W międzyczasie, atakujący odpowiada na żądania wywołane przez linki do jawnych stron, pośrednicząc w dostarczaniu zawartości autentycznej strony internetowej.

Atak *SSL Stripping* opiera się o fakt iż większość użytkowników nie jest w stanie stwierdzić różnicy w zabezpieczonym, a niezabezpieczonym przeglądaniu stron internetowych. W przypadku użytkowników potrafiących ją wskazać, atakujący może oszukać taką osobę przekierowując ją na zabezpieczoną stronę, w pełni kontrolowana przez atakującego. Jej adres zawierałby dodatkowo cały adresu docelowej strony, różniłby się jedynie znakiem bądź wykorzystany zostałby podobny znak Unicode. Pocieszającym dla potencjalnej ofiary faktem jest iż atakujący może zapewnić jej bezpieczne połączenie. Jest to jednak złudne pocieszenie, w związku z faktem iż cała komunikacja może być dowolnie modyfikowana przez atakującego. [1, Rozdz. 5, *SSL Stripping*]

4.5 Certyfikaty Man in the Middle

Atakujący, aktywnie podsłuchując komunikację pomiędzy ofiarą, a serwerem docelowym, jest w stanie ją dowolnie modyfikować. Nie jest to zadaniem trywialnym, ale jak najbardziej wykonalnym. Wyróżnić można kilka alternatywnych wektorów ataku.

Wykorzystanie podatności procesu walidacji

Bezpieczeństwo protokołu TLS zależy od poprawności sposobu walidacji danych uwierzytelniających przekazanych przez klienta. W przypadku, w którym proces walidacji nie jest poprawnie zaimplementowany, możliwym jest wykorzystanie odpowiednio sfałszowanego certyfikatu bądź łańcucha certyfikatów, który może zostać potraktowany jako poprawny.

Fałszywe certyfikaty

Uzyskanie fałszywego certyfikatu jest złożonym i skomplikowanym procesem. W tym celu, może zostać wykorzystany atak typu *brute–force*, który w przypadku 1024 bitowego prywatnego klucza certyfikatu urzędu certyfikacji, może

wymagać dużej mocy obliczeniowej. Atakujący, który posiada fałszywy certyfikat, jest w stanie pozostać niezauważonym przez większość użytkowników. Dodając do tego fakt iż w wyniku ataku Man in the Middle atakujący może ingerować w kontrole unieważnienia OCSP oraz fakt iż większość przeglądarek ignoruje awarie tego standardu, jeżeli atakujący jest w stanie przez dłuższy czas utrzymać pełną kontrolę nad połączeniem ofiary z Internetem, możliwym jest zatem iż unieważnienie fałszywego certyfikatu będzie niemożliwe.

Samo podpisany certyfikat

Stanowi najmniej wyrafinowany wektor ataku. Polega on na przedłożeniu samo podpisanego certyfikatu, którego większość pól skopiowanych jest bezpośrednio z oryginalnego dokumentu. Mimo tego iż zazwyczaj w wyniku takiego działania użytkownik otrzymuje ostrzeżenie dotyczące certyfikatu, najczęściej jest ono bez większego zastanowienia akceptowane. [1, Rozdz. 5, *MITM Certificates*]

4.6 Ostrzeżenia dotyczące certyfikatów

W celu zapewnienia najwyższego poziomu bezpieczeństwa, kryptografia wymaga uwierzytelnienia. W przypadku, w którym użytkownik nie jest w stanie stwierdzić, czy próbuje nawiązać komunikacje z pożądaną stroną, komunikacja między nimi może zostać przechwycona i kontrolowana przez potencjalnego atakującego.

Uwierzytelnianie, w kontekście protokołu TLS, odbywa się głównie za pomocą certyfikatów. Nawiązując komunikację z pożądanym serwerem o konkretnej nazwie hosta, oczekuje się iż w celu uwierzytelnienia się, przedłoży on certyfikat. W przypadku otrzymania nieważnego certyfikatu, najbardziej odpowiednią decyzją byłoby przerwanie próby nawiązania komunikacji z danym serwerem, czego przeglądarki internetowe nie wykonują. Dostawcy przeglądarek „rozwiązali" ten problem, zrzucając obowiązek na użytkowników, by w takich przypadkach wyświetlane im były ostrzeżenia dotyczące certyfikatów.

Istnieje mnóstwo niepotwierdzonych dowodów na temat przyczyn występowania nieważnych certyfikatów, aczkolwiek da się jednoznacznie określić niektóre z głównych.

Błędna konfiguracja wirtualnych hostów

Większość obecnych stron internetowych wykorzystuje port 80, który nie zapewnia szyfrowania komunikacji. Bardzo powszechnym błędem w konfiguracji jest umieszczanie stron internetowych wykorzystujących nieszyfrowany port, na tym samym adresie IP, na którym umieszczone są strony internetowe

wykorzystujące port 443. W czego wyniku, użytkownicy chcący nawiązać nieszyfrowaną komunikację przez prefiks *https*, trafiają w nieodpowiednie miejsce, w wyniku czego, certyfikat, który otrzymują nie odpowiada docelowej nazwie hosta. Część problemu stanowi stricte techniczny aspekt – nie istnieje mechanizm dla stron internetowych, który jednoznacznie byłby w stanie stwierdzić, czy obsługują one szyfrowanie. W takim przypadku, poprawnym sposobem hostowania stron wykorzystujących nieszyfrowany port 80, jest umieszczenie ich na serwerze, na którym port 443 jest zamknięty.

Niewystarczające pokrycie nazw

W niewielkiej ilości przypadków, operatorzy strony popełniają błąd, nie wskazując wszystkich wymaganych nazw hostów, które powinny zawarte być w certyfikacie. Przykładowo, mając stronę *www.example.com*, certyfikat powinien zawierać ten adres oraz zwykłą witrynę *example.com*. W przypadku innych domen wskazujących na tę stronę, powinny one również zostać zawarte w certyfikacie.

Samo podpisane certyfikaty oraz prywatne urzędy certyfikacji

Samo podpisane bądź wydane przez prywatne urzędy certyfikacji certyfikaty, nie są odpowiednie do publicznego użytku, gdyż nietrywialnym jest odróżnienie ich od certyfikatów używanych w atakach typu Man in the Middle.

Certyfikaty wykorzystywane przez urządzenia

Obecnie, większość urządzeń wykorzystuje internetowy interfejs użytkownika, wymagający bezpiecznej komunikacji. W związku z faktem iż w trakcie procesu produkcji urządzenia, nie jest znany ani jego adres IP ani nazwa hosta, oznacza to iż nie może im zostać w danej chwili przypisany ważny certyfikat. W teorii, użytkownicy końcowi mogliby instalować ważne certyfikaty samodzielnie. Jednakże w związku z faktem iż wiele urządzeń nie umożliwia wykorzystania certyfikatów dostarczonych przez użytkownika bądź urządzenia wykorzystywane są relatywnie rzadko, taka praktyka jest bardzo sporadyczna.

Wygasłe certyfikaty

Bardzo często stanowią przyczynę występowania nieważnych certyfikatów. W wielu przypadkach, operatorzy strony zapomniawszy o odnowieniu certyfikatu, nie podejmują próby jego odnowienia, rezygnując tym samym z posiadania ważnego certyfikatu.

Efektywność ostrzeżeń dotyczących certyfikatów

Priorytetem dostawców przeglądarek jest uzyskanie jak największego udziału procentowego na rynku. Niestety nie zawsze, wzrost tej wartości idzie w parze ze wzrostem poziomu bezpieczeństwa proponowanego przez daną przeglądarkę. Użytkownicy bardzo sporadycznie skarżą się na ostrzeżenia dotyczące certyfikatów, wywołanych autentycznymi atakami typu Man in he Middle, które są właściwym powodem istnienia ostrzeżeń. Dzieje się tak ze względu na fakt iż użytkownicy nie są świadomi ataków, a ostrzeżenia traktowane są przez nich jako naturalna, nieodzowna część przeglądania stron internetowych, w związku z czym, nie zgłaszają takich przypadków.

Obecnie, większość głównych przeglądarek stosuje ostrzeżenia przerywające bądź zajmujące całe okno przeglądarki. Ówcześnie, ostrzeżenie stanowiło pojedyncze okno dialogowe, które uznawane było za bardzo nieefektywną i nieskuteczną formę ostrzeżenia, ze względu na fakt iż zwykłe okno dialogowe pojawiało się również w różnych innych, mniej krytycznych przypadkach. Większość przeglądarek umożliwia użytkownikowi zaakceptowanie ostrzeżenia i przejścia dalej na stronę, co wielu z nich świadomie bądź nie czyni, mimo zagrożenia. Przeprowadzone w 2011 roku badania dotyczące skuteczności wyświetlanych ostrzeżeń dotyczących certyfikatów pokazują iż najlepszą implementację tego mechanizmu posiada przeglądarka *Mozilla Firefox*, gdzie spośród wszystkich użytkowników próbujących wejść na stronę z nieważnym certyfikatem, mimo wyświetlonego ostrzeżenia, przeszło na nią zaledwie 33% użytkowników. Znacznie gorszy rezultat osiągnęli użytkownicy przeglądarki *Chrome*, gdzie liczba takich użytkowników początkowo stanowiła 70%. W wyniku wzorowania się na projekcie przeglądarki *Firefox*, *Chrome* zdołał zredukować liczbę użytkowników przechodzących na stronę mimo ostrzeżenia do 56%.

Akceptacja ostrzeżenia, a wyjątek bezpieczeństwa

Osiągnięcie najlepszego rezultatu przez przeglądarkę *Firefox* w uprzednio wspomnianym badaniu, zawdzięcza się sposobowi, w jakim obsługuje ona przypadki, w których użytkownik próbuje wejść na stronę z nieważnym certyfikatem. Przeglądarka *Firefox* nie wykorzystuje zwykłego ostrzeżenia, które informuje jedynie o odpowiednim fakcie i wymaga jego zaakceptowania. Zamiast tego, użytkownik musi przejść kilkuetapowy proces, w którym dodaje wyjątek bezpieczeństwa dla certyfikatu danej strony. Oznacza on iż przy kolejnych wizytach, jej certyfikat będzie traktowany przez przeglądarkę jako poprawny i zaufany. Na podstawie wyników badań, można zatem wyciągnąć wnioski iż w związku z kilkoma etapami całego procesu, użytkownik przechodząc kolejne etapy zwiększa swoją świadomość o danej sytuacji i podejmuje decyzję, by jednak zaniechać odwiedzenia niezabezpieczonej strony.

Istnieje argument przeciwko rozwiązaniu omawianego problemu za pomocą

wyjątków bezpieczeństwa. Umożliwiają one wykorzystanie w prostszy sposób samo podpisanych certyfikatów. W rękach osób, które posiadają wiedzę o odpowiednim ich wykorzystaniu, nie są one właściwie niebezpieczne.

Wyjątki bezpieczeństwa dotyczące certyfikatów, użyteczne są zatem jedynie do indywidualnych potrzeb oraz dla niewielkiej grupy użytkowników technicznych, którzy wiedzą by wykorzystać ten mechanizm jedynie w przypadku bezpiecznej sytuacji oraz pewności, że użytkownicy danej strony nie będą celem ataku.

Środki łagodzące

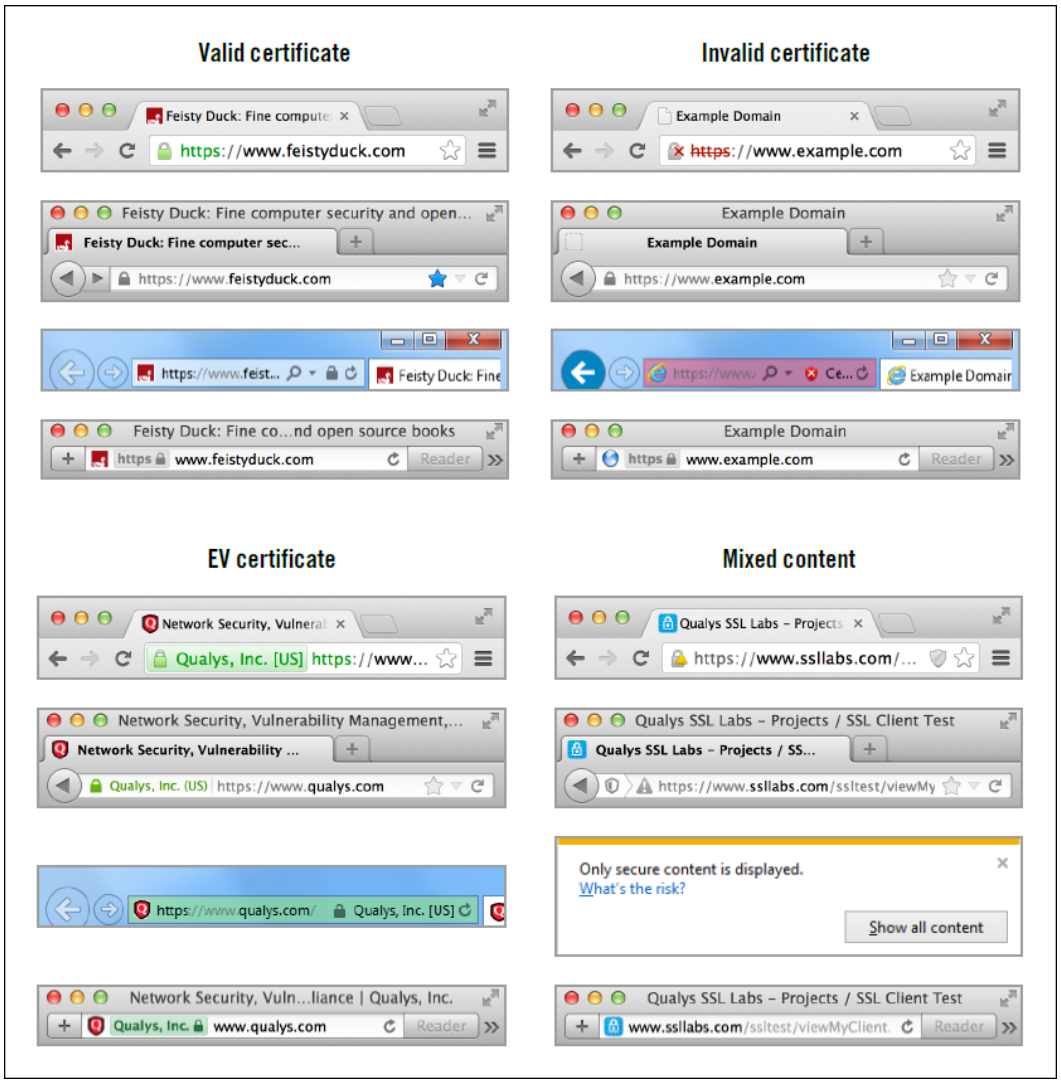
Dostosowując się do standardu HSTS można zwiększyć poziom bezpieczeństwa swojej strony. Standard ten sygnalizuje przeglądarce, by dostosowała zachowanie oraz by przyjęła bardziej rygorystyczną postawę jeżeli chodzi o szyfrowanie. Strona internetowa z wdrożonym danym standardem, posiada kolejną warstwę bezpieczeństwa, ze względu na fakt iż w przypadku problemu z certyfikatem i wyświetleniu ostrzeżenia, sytuacje takie traktowane są jako krytyczne awarie i nie mogą zostać zaakceptowane, ani pominięte przez użytkownika. [1, Rozdz. 5, *Certificate Warnings*]

4.7 Wskaźniki bezpieczeństwa

Wskaźniki bezpieczeństwa są elementami interfejsu użytkownika informującymi o zabezpieczeniach danej strony, takich jak:

- *Ta strona wykorzystuje protokół SSL*
- *Wiemy, jaki podmiot prawny zarządza tą stroną*
- *Ta strona wykorzystuje nieważny certyfikat*
- *Niektóre części tej strony nie są szyfrowane*

Przykładowe wskaźniki bezpieczeństwa przedstawia rysunek 4.2. Z wyjątkiem certyfikatu rozszerzonej walidacji (EV Certificate) łączącego podmiot prawny z daną stroną internetową, istnienie pozostałych wskaźników związane jest z faktem luźnego traktowania bezpieczeństwa przez przeglądarki oraz faktem iż szyfrowanie witryn internetowych jest opcjonalne.



Rysunek 4.2: Przykłady wskaźników zabezpieczeń [1]

Najbardziej problematyczną kwestią związaną ze wskaźnikami bezpieczeństwa stanowi fakt iż większość użytkowników nie zwraca na nie uwagi. Na podstawie badań z wykorzystaniem śledzenia ruchów gałek ocznych stwierdzono iż znaczna część użytkowników spędza bardzo małą ilość czasu na patrzenie się na pasek adresu przeglądarki, poświęcając jego resztę na skupieniu się na zawartości strony. Badania wykazały również iż żaden z użytkowników nie zauważył na danej stronie wskaźnika mówiącego o certyfikacie typu EV.

Przyczyny takiego stanu rzeczy nie są bezpośrednio wskazane. Prawdopodobnie, jednym z główniejszych czynników kształtujących taką sytuację jest brak spójności pomiędzy przeglądarkami oraz różnice pomiędzy ich wersjami. Obecnie, jedyną ze spójnych oraz wspólnych cech przeglądarek internetowych jest wykorzystanie zielonego koloru dla certyfikatów typu EV. Ta cecha jest jednak mniej respektowana wśród przeglądarek na urządzeniach mobilnych. W

związku z mniejszymi rozmiarami ich wyświetlaczy i minimalizacją interfejsu użytkownika, wskaźniki bezpieczeństwa są tam znacznie mniej wyeksponowane. [1, Rozdz. 5, *Security Indicators*]

4.8 Mieszana zawartość

Protokół TLS skupia się jedynie na pojedynczym połączeniu oraz zapewnieniu bezpieczeństwa przesyłanych w jego obrębie danych na poziomie sieci. Separacja zaimplementowana w taki sposób, najlepiej sprawdza się w przypadku prostych protokołów, na przykład SMTP. Jednak w przypadku takich protokołów jak HTTP oraz FTP, w obrębie jednego kontekstu zabezpieczeń (na przykład sesji przeglądania), może występować wiele powiązanych połączeń. W związku z faktem iż protokół TLS nie obsługuje takich sytuacji, muszą być one rozwiązywane przez programistów przeglądarek internetowych.

W przypadku szyfrowanej wersji protokołu HTTP, zatem HTTPS, trudno jest znaleźć stronę, wykorzystującą jedynie jedno połączenie. Praktycznie w przypadku wszystkich witryn, zasoby typu: znaczniki HTML, arkusze stylów, obrazki, skrypty w języku JavaScript oraz inne, dostarczane są nie tylko w wyniku wielu połączeń, ale również z wielu serwerów i innych stron internetowych. W celu poprawnego zaszyfrowania strony, koniecznym jest by cała jej zawartość pobierana była z wykorzystaniem HTTPS. W praktyce jednak, bardzo często tak nie jest, w związku z czym, taka sytuacja prowadzi do problemów dotyczących bezpieczeństwa mieszanej zawartości.

4.8.1 Główne przyczyny

W celu zrozumienia wszechobecności mieszanej zawartości, należy przyjrzeć się początkom Internetu oraz rozważyć zawrotne tempo jego rozwoju, podczas którego, zawsze skupiano się na rozwiązywaniu problemów oraz pokonywaniu ograniczeń narzuconych przez technologię, koszta oraz bezpieczeństwo.

Wydajność

W początkowych latach użytkowania protokołu SSL, wydajność szyfrowanej w ramach tego protokołu komunikacji była relatywnie słabsza, w porównaniu do jawnej, nieszyfrowanej komunikacji. Obecnie, nawet gdy serwery wyposażone są w dużo wydajniejsze procesory oraz posiadają więcej pamięci RAM, powodem do niepokoju może być szybkość wykonywanych operacji kryptograficznych. Ówcześnie, w celu uzyskania relatywnie dobrej wydajności SSL, należało wykorzystać drogie, specjalistyczne akceleratory sprzętowe. W związku z takim stanem rzeczy oraz brakiem 100% pewności, że jego wykorzystanie zapewni całkowite zabezpieczenie komunikacji, bardzo często stając przed wyborem: częściowego zabezpieczenia transmisji wymienianych danych bądź jego braku, unikano wdrażania protokołu SSL.

Obecnie, wydajność ciągle stanowi problem, aczkolwiek dotyczy on nie tyle szybkości wykonywanych operacji kryptograficznych, co opóźnień wywołanych liczną ilością wymienianych informacji między klientem, a serwerem, w celu nawiązania bezpiecznego połączenia.

Mashupy

W pewnym momencie rozwoju Internetu, narodził się koncept *mashupów*. W wyniku tego zdarzenia, strony internetowe przestały dostarczać całą prezentowaną na nich zawartość, by pewną jej część pobierać z innych i udostępniać na swojej stronie internetowej, ukrywając źródło ich pochodzenia oraz skupiając się na doświadczeniach użytkowników. W niektórych przypadkach, udostępniane treści były w pełni darmowe, natomiast w innych, działały one w ramach transakcji handlowych.

W związku z faktem iż *mashupy* są realizowane głównie za pomocą kodu JavaScriptowego z zewnętrznej strony, daje to im możliwość przejęcia prawie pełnej kontroli nad stronami udostępniającymi daną zawartość. Takie rozwiązanie zatem, umożliwia znaczne obniżenie kosztów, powodując przy tym również problem dla użytkowników. W sytuacji, w którą na danej witrynie internetowej, zaangażowana jest tak duża ilość stron, może być trudnym dla potencjalnego użytkownika zrozumienie, z którą z nich właściwie się komunikuje i która z nich przechowuje jego dane. Kolejny problem powstający przy wykorzystywaniu *mashupów* dotyczy szyfrowania. Strony internetowe udostępniające zawartość z innych witryn, w wielu przypadkach udostępniają ją przez niezaszyfrowany kanał komunikacyjny. Często z powodu oszczędności, w związku z faktem iż udostępnianie danej zawartości wykorzystując szyfrowanie, jest zazwyczaj opcją dostępną po uiszczeniu opłaty w odpowiedniej wysokości. W wyniku takiego działania, udostępniające w taki sposób zawartość strony internetowe, niezgodnie z prawdą poświadczają o bezpieczeństwie komunikacji.

Koszty infrastruktury

Wraz ze wzrostem konkurencji między stronami internetowymi, dostarczanie witryn z jednej geograficznej lokalizacji oraz pozostanie przy tym konkurencyjnym, stało się niemożliwe. Wzrosła również popularność sieci dostarczania treści (CDN)[6], która wykorzystywana była w celu dostarczania treści przy najlepszej możliwej wydajności. Koncept polegał na tym, by rozpowszechniwszy wiele serwerów w różnych zakątkach świata, użytkownik chcąc odwiedzić witrynę, połączył się do najszybszego z serwerów.

Problem z sieciami CDN polega na tym iż z założenia, powinny one udostępniać bardzo duże ilości zazwyczaj statycznych danych, równie dużej, a nawet większej ilości użytkowników. W związku z tym, generowane są duże koszty wydajnościowe. Intensywne operacje kryptograficzne zwiększają zużycie procesora i pamięci RAM oraz mogą wpływać na buforowanie i obciążać

zarządzanie kluczami oraz certyfikatami. Oprócz problemu dotyczącego operacji kryptograficznych, istnieje również problem dotyczący adresów IP. Dla jawnej, nieszyfrowanej komunikacji poprzez protokół HTTP, dla której wirtualny hosting stron internetowych jest szeroko wspierany, adresy IP nie mają znaczenia. Taki stan rzeczy sprawia iż hosting oraz dystrybucja na dużą skalę, odbywa się w prosty sposób. Inaczej przedstawia się sytuacja w przypadku bezpiecznych publicznych stron, dla których wirtualny hosting nie jest możliwy. Oznacza to konieczność implementacji mapowania stron internetowych na adresy IP i tym samym na serwery, co znacznie komplikuje architekturę oraz zwiększa ilość narzutów.

Generalnie, integralność szyfrowania na poziomie strony, była przez przeglądarki zapewniana w niewielkim stopniu. Kwestie dotyczące mieszanych treści zostały dozwolone, a następnie głęboko zakorzenione w kulturze rozwoju.

4.8.2 Wpływ

Wpływ zagadnień dotyczących treści mieszanych zależy od natury niezabezpieczonych zasobów. Z upływem lat, utworzyły się dwa nowe terminy. Pierwszy z nich – pasywna mieszana zawartość (*miexed passive content*) – odnosi się do zasobów o niższym ryzyku, takich jak obrazy. Natomiast drugi z nich – aktywna mieszana zawartość (mixed active content) – odnosi się do zasobów o podwyższonym ryzyku, takich jak skrypty języka JavaScript czy znaczniki HTML. Druga z wymienionych kategorii może stanowić poważne zagrożenie. Wykonanie pojedynczego, niechronionego włączenia (inkluzji) piku Javascriptowego, może zostać przechwycone przez aktywnego atakującego, by następnie mogło zostać wykorzystane w celu przejęcia pełnej kontroli nad stroną i wykonania na niej dowolnych operacji, z wykorzystaniem tożsamości ofiary. Przypadek najmniej niebezpieczny, choć mogący prowadzić do pishingu, przewiduje wysyłanie do ofiary przez atakującego wiadomości, osadzonych w obrazach. Możliwym jest również wstrzykiwanie exploitów, wymierzając je w przeglądarkowe mechanizmy przetwarzania obrazów. Niektóre przeglądarki wykorzystują mechanizm sniffowania treści, który jest w stanie przetworzyć obraz jako skrypt. W tym przypadku, atakujący jest również w stanie przejąć kontrolę nad stroną.

4.8.3 Powszechność mieszanych treści

W 2011 roku, firma Qualys przeprowadziła badania dotyczące powszechności mieszanych treści oraz kilku innych problemów z poziomu aplikacji, których wynikiem było pełne złamanie szyfrowania aplikacji webowych. Badaniu zostało poddanych około 250 tysięcy „bezpiecznych" stron internetowych z pierwszej listy rankingu Alexa.[2]. Wyniki badań ukazywały iż 22.41% stron

wykorzystywało niezabezpieczoną zawartość. W przypadku, w którym z mieszanych treści wyłączone zostały obrazy, liczba ta spadła i stanowiła już tylko 18.71%.

Kolejne, bardziej szczegółowe badania 18 526 stron wyłonionych spośród najwyżej uplasowanych 100 tysięcy stron z rankingu Alexa odbyło się w 2013 roku. Dla każdej witryny internetowej, przeanalizowano do maksymalnie 200 bezpiecznych stron, dając sumę 481 656 stron. Wyniki badań przedstawia rysunek 4.3.

	# Inclusions	% remote	# Files	# Webpages	% Websites
Image	406,932	38%	138,959	45,417	30%
Frame	25,362	90%	15,227	15,419	14%
CSS	35,957	44%	6,680	15,911	12%
JavaScript	150,179	72%	29,952	45,059	26%
Flash	1,721	62%	638	1,474	2%
Total	620,151	47%	191,456	74,946	43%

Rysunek 4.3: Zestawienie mieszanych treści na 481 656 stronach poddanych badaniu [1]

4.8.4 Środki łagodzące

Mimo dosyć luźnego podejścia przeglądarek do kwestii dotyczących mieszanej zawartości, odpowiednio wdrożona strona internetowa może być odporna na ataki. W celu minimalizacji ryzyka ataku nawet źle wdrożonych witryn, mogą zostać wykorzystane dwie technologie eliminujące problemy z mieszaną zawartością.

Pierwszą z nich jest mechanizm HSTS (*HTTP Strict Transport Security*), który wymusza bezpieczne pobieranie zasobów, mimo przypadków, w których użytkownik popełnia błąd – na przykład próbując uzyskać dostęp do strony na porcie 80 – oraz mimo przypadków, w których wystąpiły błędy w implementacji – na przykład, gdy programiści bezpiecznej strony, umieszczają na niej niezabezpieczony link). Kolejnym mechanizmem jest CSP (*Content security policy*), który blokuje próby pozyskania zasobów z niezabezpieczonych zewnętrznych stron.

Zarówno HSTS, jak i CSP jest mechanizmem deklaratywnym, zatem może być on dodany na poziomie serwera, bez konieczności zmiany aplikacji. W pewnym sensie, oba mechanizmy można traktować jak „siatki bezpieczeństwa", ze względu na fakt iż mogą one je zapewnić, nawet w przypadku nieprawidłowej implementacji witryny. [1, Rozdz. 5, *Mixed Content*]

4.9 Certyfikaty rozszerzonej walidacji

Certyfikaty typu EV stanowią specjalną klasę certyfikatów, wiążących nazwę domeny z danym podmiotem prawnym. Oferują one dwie główne zalety. Pierwsza z nich dotyczy tego iż tożsamość właściciela domeny jest znana i zaszyfrowana w certyfikacie. Drugą z zalet stanowi manualny proces ich weryfikacji, co czyni sfałszowanie certyfikatu tego typu nietrywialnym.

Z drugiej strony, dyskusyjną kwestią jest, czy te zalety niosą jakiekolwiek praktyczne korzyści, w odniesieniu do użytkowników i ich ogólnej populacji. W związku z faktem iż użytkownicy bardzo rzadko zwracają uwagę na podstawowe wskaźniki bezpieczeństwa oraz te, świadczące o posiadaniu przez daną domenę certyfikatu EV, bardzo prawdopodobnym jest iż nie zwrócą również uwagi na powiązanie wskazujące na właściciela nazwy domeny. Ponadto, fałszywe certyfikaty typu DV mogą być wykorzystywane w celu ataku stron posiadających certyfikat EV. Jedynym sposobem, stanowiącym zapobiegnięcie tego typu atakom jest edukacja użytkowników końcowych na temat certyfikatów EV. Zatem, uświadomienie im, czym są tego typu certyfikaty, wskazanie dobrych nawyków do zwracania uwagi na wskaźniki bezpieczeństwa i ostrzeżenia związane z certyfikatami. Taka wizja wydaje się mało prawdopodobna, biorąc pod uwagę odsetek użytkowników, którzy mimo ostrzeżeń przechodzą do zabezpieczonej strony.

Kolejnym problemem dotyczącym certyfikatów EV stanowi fakt iż są one wykrywane oraz wskazywane na poziomie strony, bez uwzględnienia typu certyfikatu użytego przez zasoby. Biorąc pod uwagę wysoki koszt certyfikatów typu EV, nie jest zjawiskiem niecodziennym wykorzystywanie przez złożone strony certyfikatów DV, dla poddomen niewidocznych w dużej mierze. Fakt ten oznacza iż ostrożny atakujący sieci, może korzystać certyfikatu typu DV na stronie posiadającej certyfikat EV, potencjalnie, bez wpływu na zielone wskaźniki bezpieczeństwa.

Zasoby dostarczone z zewnętrznych nazw domen

W wielu przypadkach, strony będą wykorzystywały certyfikat EV na swojej głównej stronie, ale wiele zasobów pozyskiwanych będzie z wielu innych nazw hostów, które zazwyczaj wykorzystują certyfikaty typu DV. Połączenia przeglądarki dla innych nazw hostów, mogą być zatem przechwytywane za pomocą fałszywego certyfikatu DV, co następnie może prowadzić do wstrzyknięcia złośliwego oprogramowania.

Kradzież plików cookies

W związku z faktem iż przeglądarki nie wymuszają ciągłości certyfikatu, możliwym jest wykorzystanie certyfikatu typu DV do przechwycenia połączenia z główną nazwą domeny, w celu kradzieży bądź zastąpienia ciasteczek nowy-

mi, a następnie przekierowania z powrotem do docelowego serwera. Sprawnie wykonany atak tego typu, może zostać niezauważony przez większość użytkowników.

Trwałe wstrzyknięcie złośliwego oprogramowania

Szkodliwe oprogramowanie, które zostało wstrzyknięte do przeglądarki, może być przechowywane w pamięci podręcznej plików, w przypadku, w którym wymuszone jest buforowanie. Tak wstrzyknięty malware może pozostać aktywny przez długi czas, nawet przy kolejnych odwiedzinach danej strony. [1, Rozdz. 5, *Extended Validation Certificats*]

Rozdział 5

Aktualizacja serwera math

Z natury rzeczy, zabezpieczanie serwera nie jest jednorazową akcją lecz permanentnym procesem. Administrator powinien śledzić doniesienia o wykrytych podatnościach w uruchomionym na serwerze oprogramowaniu oraz reagować na bieżąco poprzez dostosowywanie konfiguracji, instalowanie poprawek bezpieczeństwa oraz aktualizacji. Ze względu na fakt iż bezpieczeństwo systemu i wygoda użytkowników to przeciwległe bieguny, administrator jest skazany na ciągłe szukanie kompromisów.

W przypadku serwera **math**, z biegiem czasu, skumulowało się kilka problemów związanych z bezpieczeństwem systemu. Nie da się ukryć, że oprogramowanie na tym serwerze mocno się zestarzało, w związku z czym, pilnej aktualizacji wymagały następujące komponenty:

1. biblioteka OpenSSL,
2. serwer SSH
3. serwer HTTP,
4. serwer IMAP i POP3.

Przygotowanie, przetestowanie i uruchomienie nowych wersji tych składników systemu stanowi część praktyczną niniejszej pracy.

5.1 Procedura aktualizacji

Z uwagi na fakt iż na serwerze **math**, niemal nieprzerwanie od instalacji w 2013 roku, pracuje nieco wiekowy już Solaris 10, zadanie aktualizacji wymienionych komponentów nie jest trywialne. Na danej platformie nie istnieje system podobny do *apt-get*, czy *yum* znany z Linuxa. Jednakże nawet w przypadku istnienia takiego systemu, automatyczna aktualizacja na serwerze wymaga wstępnych testów i sprawdzenia. Proces aktualizacji jest dość pracochłonny. Można w nim wyróżnić następujące etapy:

1. kompilacja kodu źródłowego,
2. przygotowanie pakietu instalacyjnego,
3. instalacja, konfiguracja i testowanie w środowisku deweloperskim,
4. instalacja i uruchomienie oprogramowania na serwerze.

W przypadku, w którym testy wypadną pomyślnie, wszystko powiedzie się zgodnie z planem i uzyskamy satysfakcjonujące wyniki, można przejść do ostatniego punktu. W przeciwnym razie należy wrócić do odpowiedniego punktu, by poprawić kompilację lub pakiet instalacyjny. Niejednokrotnie zdarza się powtarzać 3 pierwsze punkty wiele razy, tym częściej im bardziej złożone jest przygotowywane oprogramowanie.

Zanim przejdziemy do omówienia bardziej szczegółowo jak przebiegały prace, w przypadku każdego z wymienionych wcześniej programów, wyjaśnijmy kilka ogólnych kwestii na wszystkich etapach.

5.1.1 Kompilacja

Środowiskiem deweloperskim jest maszyna z zainstalowaną taką samą wersją systemu operacyjnego i z takim samym zestawem bibliotek jak na serwerze. Zawartość pliku `/etc/release` zdradza szczegóły wersji systemu Solaris. W naszym wypadku mamy:

```
Solaris 10 10/09 s10x_u8wos_08a X86
```

z zainstalowanym kompilatorem:

```
Sun C 5.12 SunOS_i386 2011/11/16
```

Docelowo interesuje nas platforma 64 bitowa. Ponieważ jednak są jeszcze w użyciu starsze maszyny 32 bitowe, kompilujemy wszystko również w tej wersji.

Sama kompilacja to także proces kilkuetapowy. Zaczynamy od przygotowania środowiska do kompilacji. Proces ten polega na odpowiednim ustawieniu zmiennych środowiskowych używanych przez zastosowany w konkretnym przypadku system wspomagający kompilację kodu. Najszerzej rozpowszechniony jest *GNU build system* oparty na `autoconf`, `automake` i `libtool`. Aczkolwiek, twórcy serwera HTTP Apache używają jego nieco zmodyfikowaną wersję. Najbardziej podstawowe zmienne to:

`CC` – ścieżka do kompilatora C,

`CFLAGS` – zestaw flag i opcji kompilatora C,

`CPPFLAGS` – zestaw flag i opcji preprocesora C,

`CXX` – ścieżka do kompilatora C++,

`CXXFLAGS` – zestaw flag i opcji preprocesora C++,

`LDFLAGS` – zestaw flag i opcji programu konsolidującego `ld`

W naszym przypadku, do kompilacji w wersji 64 bitowej, przypisujemy następujące wartości tym zmiennym:

```
CC=cc
CFLAGS="-fast -xtarget=pentium4 -m64 -KPIC
  -xregs=no%frameptr -xstrconst -Xa -mt
  -I/opt/cfw/include -I/usr/sfw/include -I/opt/sfw/include"
CPPFLAGS="-I/opt/cfw/include -I/usr/sfw/include -I/opt/sfw/include"
CXX=CC
CXXFLAGS="-fast -xtarget=pentium4 -m64 -KPIC -mt
  -I/opt/cfw/include -I/usr/sfw/include -I/opt/sfw/include"
LDFLAGS="-L/opt/cfw/lib/64 -R/opt/cfw/lib/64 -L/usr/sfw/lib/64
  -R/usr/sfw/lib/64 -L/opt/sfw/lib/64 -R/opt/sfw/lib/64"
```

W wersji 32 bitowej usuwamy przełącznik `-m64` oraz podkatalog `64` ze ścieżek w `LDFLAGS`.

Kolejny etap to konfiguracja różnych parametrów wymaganych do poprawnej kompilacji. Aby dany pakiet oprogramowania mógł być przeniesiony na różne platformy, kod źródłowy musi być odpowiednio dostosowany. Często w kodzie znajduje się duża ilość różnych przełączników `#if`/`#endif`, do których flagi ustalane są właśnie podczas tej konfiguracji. Poza tym, ustalane są parametry wymagane przez konkretny kompilator, wykrywana jest lokalizacja plików nagłówkowych i bibliotek bo na każdej platformie mogą one znajdować się w różnych miejscach.

Typowa konfiguracja pakietu oprogramowania polega na uruchomieniu skryptu `configure` w głównym katalogu, gdzie znajdują się źródła. Skrypt ten jest automatycznie generowany przez autorów oprogramowania programem `autoconf`, przed publikacją pakietu ze źródłami. Zawsze warto najpierw uruchomić ten skrypt z opcją `--help`, aby zobaczyć jakie parametry kompilacji ewentualnie możemy dostosować do własnych potrzeb. Zawsze podajmy opcję:

```
--prefix=/opt/cfw
```

określającą miejsce instalacji oprogramowania, czyli katalog `/opt/cfw`. Nazwa CFW pochodzi od Community FreeWare. Powstała ona dzięki aktywnym użytkownikom Solaris w latach 2003-2005, gdy firma Sun Microsystems planowała zakończenie wsparcia dla platformy x86.

W trakcie konfiguracji źródeł należy liczyć się z możliwością napotkania pewnych trudności, na przykład ze znalezieniem wymaganych plików nagłówkowych lub bibliotek. Wystarczy wtedy na ogół podać ścieżki używając odpowiednich opcji skryptu `configure`. Często zdarza się, że nowa wersja kompilowanego oprogramowania wymaga nowszej wersji pewnych bibliotek, które już mamy. Wtedy niestety musimy odłożyć rozpoczętą kompilację i skompilować nowsze, wymagane biblioteki.

Jeśli etap konfiguracji źródeł przejdzie pomyślnie, to przechodzimy do właściwego etapu kompilacji uruchamiając program `make`. Na platformie Solaris oryginalny `make` ma problemy z wieloma rozszerzeniami używanymi w systemie *GNU build system* i dlatego używamy właśnie wersji GNU dostępnej jako `gmake`. Na maszynie wyposażonej w kilka rdzeni warto użyć opcję `-j` podając liczbę zadań (ang. jobs), do równoległego wykonania. Może to znacznie skrócić czas potrzebny na skompilowanie całego pakietu.

Na tym etapie, na starszej i mało popularnej platformie jaką jest Solaris, nader często kompilacja przerywana jest i sygnalizowane są problemy. Mogą to być nieprawidłowo dobrane parametry kompilatora, niewłaściwie dobrane ścieżki do plików nagłówkowych lub bibliotek, ale bywa też, że konieczne są zmiany w kodzie źródłowym programu. W efekcie, zdarza się, że cały proces kompilacji należy zacząć od nowa, czyli od dostosowania środowiska.

Ustawienia środowiska i wszystkie dokonane zmiany w kodzie warto sobie zanotować, albo nawet skorzystać z narzędzi `diff` i `patch`, aby na przyszłość uniknąć powtarzania dochodzenia, co gdzie należy poprawić, aby uzyskać zadowalający efekt.

5.1.2 Pakiet instalacyjny

Kompilując nowe oprogramowanie lub jego nowszą wersję warto poświęcić nieco więcej czasu i dodatkowo przygotować pakiet instalacyjny. Ułatwi to ewentualną deinstalację, a co najważniejsze, łatwo będzie można wykonać instalację na innych maszynach. Opracowane wcześniej pakiety instalacyjne, również do starszych wersji Solaris, znajdą się na stronie:

`http://math.uwb.edu.pl/ftp/solaris/x86/`

W podkatalogu `5.10/64` znajdują się najnowsze pakiety. Nazwa 64 oznacza, że zawierają one kod 64 bitowy i 32 bitowy wybierany automatycznie podczas uruchamiania programów w zależności od platformy sprzętowej. Wybór dokonywany jest za pomocą programu `isaexec`, do którego prowadzą dowiązania symboliczne odpowiadające wykonywalnym programom. Rzeczywiste binaria umieszczone są odpowiednio w katalogach `i86` oraz `amd64`, odpowiednio dla wersji 32 i 64 bitowych programu. Listę możliwych natywnych zestawów instrukcji wykonywalnych na danej platformie Solaris można sprawdzić programem `isalist`. Dla procesora Intel Core i7 3820 wyświetlane jest:

```
amd64 pentium_pro+mmx pentium_pro pentium+mmx pentium i486 i386 i86
```

Pakiet instalacyjny, nie wdając się nadmiernie w szczegóły techniczne, tworzymy w następujący sposób (dokumentacja systemu `pkg` w Solaris [3]).

Zaczynamy od przygotowania metadanych. Tworzymy katalog o nazwie takiej jak budowany pakiet. Tworzymy w nim plik `pkginfo`. Znajdują się tutaj między innymi informacje takie jak: nazwa pakietu, nazwa zawartego w nim oprogramowania, jego numer wersji, krótki opis, adres strony z której pobrano źródła, miejsce instalacji w systemie. W naszym przypadku tym miejscem jest katalog `/opt/cfw` i względem niego będą ustalane ścieżki plików znajdujących się w pakiecie, poza tymi, dla których podamy położenie bezwzględne. Kolejny istotny plik to `depend` zawierający listę pakietów od których zależy nasz przygotowywany pakiet. Będą to różne wymagane biblioteki. Opcjonalnie można przygotować skrypty `preinstall`, `postinstall`, `preremove`, `postremove` wykonywane w odpowiednim momencie instalacji pakietu, co jednoznacznie sugerują ich nazwy.

Teraz możemy przygotować katalog docelowy instalacji `/opt/cfw`. Jest to link symboliczny do wcześniej utworzonego, pustego katalogu tymczasowego `basedir` w miejscu, w którym budujemy pakiet instalacyjny. Wołamy `gmake install` uzyskując potrzebny układ plików i katalogów do instalacji.

Następny krok to przygotowanie katalogu zawierającego pliki do zainstalowania w systemie. Ich położenie po instalacji określone jest w `pkginfo`. Zwykle w takim katalogu mamy typowe dla UNIXa podkatalogi zwierające odpowiednie składniki oprogramowania:

`bin` – wykonywalne programy,

`etc` – pliki konfiguracyjne,

`include` – pliki nagłówkowe,

`lib` – biblioteki,

`man` – instrukcje obsługi,

`sbin` – wykonywalne programy administracyjne,

`share` – współdzielone pliki, niezależne od platformy.

Budowę pakietu wykonujemy przy pomocy programów zawartych w systemie. Program `pkgproto` tworzy pełną listę plików, które wejdą w skład pakietu, w tym także metadanych. Następnie uruchamiamy `pkgmk`, który na podstawie wcześniej opracowanej listy tworzy pakiet w postaci odpowiedniej struktury plików. Taki pakiet jest już gotowy do instalacji. W tej postaci nie jest jednak zbyt wygodny do przenoszenia. Programem `pkgtrans` przekształcamy go w strumień danych (ang. datastream format), czyli pojedynczy plik, który warto skompresować za pomocą programu `gzip`.

5.1.3 Testowanie

Nawet jeśli wykonaliśmy pracowicie wszystkie kroki kompilacji i mamy już pakiet instalacyjny, przed instalacją na maszynie produkcyjnej, warto przetestować wszystko w podobnym środowisku. Bywa, że drobne przeoczenie podczas kompilacji lub budowy pakietu instalacyjnego wymaga powtórzenia niemal całej procesu. W przypadku, w którym jest to aktualizacja na serwerze produkcyjnym, dana sytuacja stanowi poważny problem, ze względu na przestój w dostawie usług bądź konieczności przywrócenia poprzedniej wersji.

Typowe problemy, na które warto zwrócić uwagę:

- Czy przy uruchamianiu programów prawidłowo podłączane są wymagane biblioteki? Często zdarza się, że flagi `-R` mówiące o położeniu bibliotek w czasie wykonywania (ang. runtime linking) zostały zignorowane lub zmienione przy kompilacji. Wtedy niestety, koniecznym jest powtórzenie kompilacji.

- Czy wszystkie zainstalowane pliki są na swoim miejscu oraz mają prawidłowo ustawione prawa dostępu, właściciela i grupę? Złe ustawienia powodują, że program nie ma dostępu do loga, pliku konfiguracyjnego itp. Na ogół wystarczy wówczas poprawić pakiet instalacyjny.

- Czy programy działają w oczekiwany sposób? Jest to najtrudniejszy do zweryfikowania aspekt. Podczas testów jesteśmy w zasadzie w stanie wykluczyć tylko dość oczywiste problemy. Czasem konieczna jest ponowna kompilacja, czasem to kwestia poprawienia konfiguracji i aktualizacji pakietu instalacyjnego.

Testy wykonywane były na wirtualnej maszynie, na którą przeniesiono kopię systemu z serwera **math** z pominięciem danych jak poczta użytkowników, bazy danych, strony WWW. Pozwoliło to bardzo dokładnie odwzorować warunki na maszynie produkcyjnej i wyłapać wszelkie nieprawidłowości na każdym etapie aktualizacji oprogramowania.

5.1.4 Instalacja i uruchomienie

Po ukończeniu testów i ewentualnych poprawkach jesteśmy gotowi do przeprowadzenia aktualizacji na serwerze **math**. Wyczerpujące eksperymenty w środowisku testowym znacznie ułatwiły proces instalacji nowszej wersji oprogramowania i jego konfiguracji. Wszystko przebiegło bardzo sprawnie, i co najważniejsze, zgłoszona została bardzo mała liczba uwag, ze strony użytkowników.

5.2 OpenSSL

Biblioteka **OpenSSL** jest dynamicznie linkowana zarówno przez oprogramowanie serwera **OpenSSH** jak i serwera **Apache**. Dlatego jej aktualizacja musi być wykonana jako pierwsza. W momencie kompilacji najnowsza wersja **OpenSSL** miała numer `1.0.2n`.

5.2.1 Kompilacja

Zaczynamy od uruchomienia skryptu `Configure` przygotowującego środowisko do kompilacji **OpenSSL** na naszej platformie. Jest to skrypt Perlowy. Modyfikujemy w nim opcje dla platform:

- `solaris-x86-cc`
- `solaris64-x86_64-cc`

podając typowe wartości jak w `CFLAGS`. W pierwszym przypadku:

```
-fast -xtarget=pentium4 -KPIC -xregs=no%frameptr -xstrconst -Xa -mt
```

natomiast w drugim:

```
-fast -xtarget=pentium4 -m64 -KPIC -xregs=no%frameptr -xstrconst
  -Xa -mt -DL_ENDIAN
```

W katalogu, gdzie rozpakowaliśmy źródła uruchamiamy:

```
./Configure solaris64-x86_64-cc \
   --prefix=/opt/cfw \
   --openssldir=/opt/cfw/etc/openssl \
   threads shared
```

Opcja `--openssldir` określa katalog konfiguracyjny dla oprogramowania **OpenSSL**. Następnie uruchamiamy kompilację i przygotowujemy pakiet w wersji 64 bitowej. Powtarzamy operację kompilacji, tym razem w wersji 32 bitowej. Zatem:

```
gmake clean
./Configure solaris-x86-cc \
   --prefix=/opt/cfw \
   --openssldir=/opt/cfw/etc/openssl \
   threads shared
gmake
```

i teraz przygotowujemy pakiet w wersji 32 bitowej.

5.2.2 Pakiet instalacyjny

Plik `pkginfo` zawierający metadane pakietu:

```
PKG=CFWopenssl
NAME=OpenSSL - Secure Socket Layer
ARCH=i386
VERSION=1.0.2n
CATEGORY=system
DESC=Open Source toolkit implementing the SSL v2/v3 and TLS v1 protocols
  as well as a general purpose cryptography library
WEBSITE=http://www.openssl.org/
PSTAMP=Dec 18, 2017
BASEDIR=/opt/cfw
```

5.2.3 Instalacja

W trakcie testów sprawdziliśmy, że po aktualizacji biblioteki **OpenSSL** wszystkie programy jej używające działają poprawnie. Sama instalacja biblioteki **OpenSSL** jest dość trywialna. Należy pamiętać, aby przed deinstalacją starej wersji wyłączyć wszystkie zależne od niej usługi:

```
gzcat -d openssl-1.0.2n-Solaris10-x86.pkg.gz > /tmp/openssl
svcadm disable -s apache mda sendmail
pkgrm CFWopenssl
pkgadd -d /tmp/openssl
svcadm enable apache mda sendmail
rm /tmp/openssl
```

Opcja `-s` spowoduje, że `svcadm` wyłączy kolejne usługi czekając na ich prawidłowe zamknięcie.

5.3 OpenSSH

Oprogramowanie **OpenSSH** zapewnia bezpieczne połączenie klienta z uruchomioną na serwerze usługą **SSH**. W zestawie mamy zarówno klienta jak i serwer. Na serwerze **math** znajduje się zmodyfikowana przez firmę Sun Microsystems wersja **OpenSSH** oprogramowanie zwana SunSSH. Aktualizacja będzie zatem polegać na odinstalowaniu SunSSH i zainstalowaniu nowego pakietu **OpenSSH**. W momencie kompilacji najświeższa, dostępna jest wersja o numerze `7.6`.

5.3.1 Kompilacja

Przygotowujemy środowisko do kompilacji ustawiając zmienne środowiskowe `CFLAGS`, `LDFLAGS` itp. dla wersji 64 bitowej i uruchamiamy:

```
./configure --prefix=/opt/cfw \
  --libexecdir=/opt/cfw/sbin \
  --sysconfdir=/opt/cfw/etc/ssh \
  --mandir=/opt/cfw/man \
  --disable-static \
  --with-pam  \
  --with-mantype=man \
  --with-pid-dir=/var/run \
  --with-lastlog=/var/adm/lastlog \
  --with-privsep-user=ssh \
  --with-privsep-path=/var/cfw/ssh \
  --without-ssh1
```

Znaczenie poszczególnych opcji jest następujące:

`--libexecdir` – lokalizacja programów pomocniczych

`--sysconfdir` – lokalizacja plików konfiguracyjnych

`--mandir` – lokalizacja instrukcji obsługi

`--disable-static` – wyłączamy tworzenie bibliotek statycznych

`--with-pam` – włącza wsparcie PAM (Pluggable Authentication Modules)

`--with-mantype` – określa typ generowanych instrukcji obsługi

`--with-pid-dir` – lokalizacja pliku zawierającego identyfikator procesu

`--with-lastlog` – lokalizacja dziennika kto logował się do systemu

`--with-privsep-user` – nazwa użytkownika o obniżonych przywilejach

`--with-privsep-path` – lokalizacja, gdzie będą wykonywane operacje o obniżonych przywilejach

`--without-ssh1` - wyłączamy wersję protokołu SSH 1

W trakcie kompilacji okazało się, że do zmiennej `LD` w głównym `Makefile` należy dodać `$(CFLAGS)`. Jest to konieczne, gdy do konsolidacji używa się pośrednio kompilatora `cc` zamiast programu konsolidującego `ld`.

5.3.2 Pakiet instalacyjny

Plik `pkginfo` zawierający metadane pakietu:

```
PKG=CFWopenssh
NAME=OpenSSH - SSH client, server and utilities
ARCH=i386
VERSION=7.6
CATEGORY=application
DESC=The premier connectivity tool for remote login with the SSH
  protocol.
WEBSITE=https://openssh.com/
EMAIL=mariusz@solaris-x86.org
PSTAMP=Mariusz Zynel, Dec  18, 2017
BASEDIR=/opt/cfw
```

Do poprawnego działania nowej usługi SSH wymagane jest wygenerowanie kluczy kryptograficznych oraz dodanie do systemu dodatkowej grupy `ssh` i użytkownika `ssh`. Aby proces ten zautomatyzować i nie musieć o tym pamiętać przy każdej instalacji pakietu, utworzyliśmy następujący skrypt `postinstall`:

```
# OpenSSH postinstall script
#
# Copyright 2017 Marcin Roszkowski & Mariusz Żynel

/lib/svc/method/cfw/sshd -c

if grep "^ssh:" $PKG_INSTALL_ROOT/etc/group > /dev/null
then
    exit 0
fi

cat >> $PKG_INSTALL_ROOT/etc/group << EOF
ssh::110:
EOF

cat >> $PKG_INSTALL_ROOT/etc/passwd << EOF
ssh:x:110:110:SSH Server:/var/cfw/ssh:/bin/false
EOF

cat >> $PKG_INSTALL_ROOT/etc/shadow << EOF
ssh:*LK*:::::::
EOF

exit 0
```

Dodanie samej usługi SSH do systemu po instalacji pakietu wymaga utworzenia tak zwanego *manifestu* tej usługi, zatem pliku XML o odpowiedniej strukturze. Wykorzystaliśmy tutaj manifest z pakietu SunSSH. Dostosowanie polegało na zmianie lokalizacji plików. Nazwę usługi zostawiliśmy bez zmian `svc:/network/ssh`. Nowy manifest, wraz z skryptami: instalacyjnym

`i.manifest` oraz deinstalacyjnym `r.manifest`, dołączamy do pakietu. Docelowo zostanie on umieszczony w `/var/svc/manifest/cfw/network/ssh.xml`. Z manifestem instalowany jest skrypt startowy: `/lib/svc/method/cfw/sshd`. W tym wypadku także wykorzystaliśmy odpowiedni skrypt z pakietu SunSSH modyfikując w nim wyłącznie lokalizacje plików.

Aby ułatwić uruchomienie **OpenSSH** z nowego pakietu dostosowaliśmy ścieżki w pliku konfiguracyjnym `/opt/cfw/etc/ssh/sshd_config`. Dostosować należało lokalizację kluczy i podsystemu `sftp`.

5.3.3 Testowanie

Problematyczną kwestię stanowiło pytanie dotyczące możliwości reinstalacji oprogramowania SSH, z wykorzystaniem właśnie połączenia SSH. Wprawdzie istnieje możliwość podłączenia monitora i klawiatury do serwera **math** w serwerowni, aczkolwiek chcieliśmy tego uniknąć. Okazało się, że jest to wykonalne bez właściwie znacznie skomplikowanych zabiegów. W momencie kiedy jesteśmy zalogowani do systemu przez SSH w pamięci rezyduje proces serwera `sshd`. W tym czasie z systemu plików możemy usunąć wszystkie składniki tego procesu. Nie będą one widoczne, ale fizycznie zostaną usunięte dopiero, gdy proces się zakończy.

5.3.4 Instalacja i uruchomienie

Instalacja miała miejsce 8 stycznia 2018. Zaczęliśmy od wyłączenia usługi SSH i deinstalacji starego oprogramowania SSH:

```
svcadm disable -s ssh
pkgrm SUNWsshu SUNWsshr SUNWsshdu SUNWsshdr SUNWsshcu
```

Instalacja nowego pakietu polegała na wykonaniu następujących poleceń:

```
gzcat -d openssh-7.6-Solaris10-x86.pkg.gz > /tmp/openssh
pkgadd -d /tmp/openssh
rm /tmp/openssh
```

Po instalacji, w pliku `/opt/cfw/etc/ssh/sshd_config` zmodyfikowany został port, na którym nasłuchuje serwer SSH, ze standardowego 22 na 9022, aby utrudnić do niego dostęp potencjalnym atakującym. Następnie uruchomiona została usługa SSH:

```
svcadm enable ssh
svcs -x
svcs -p ssh
```

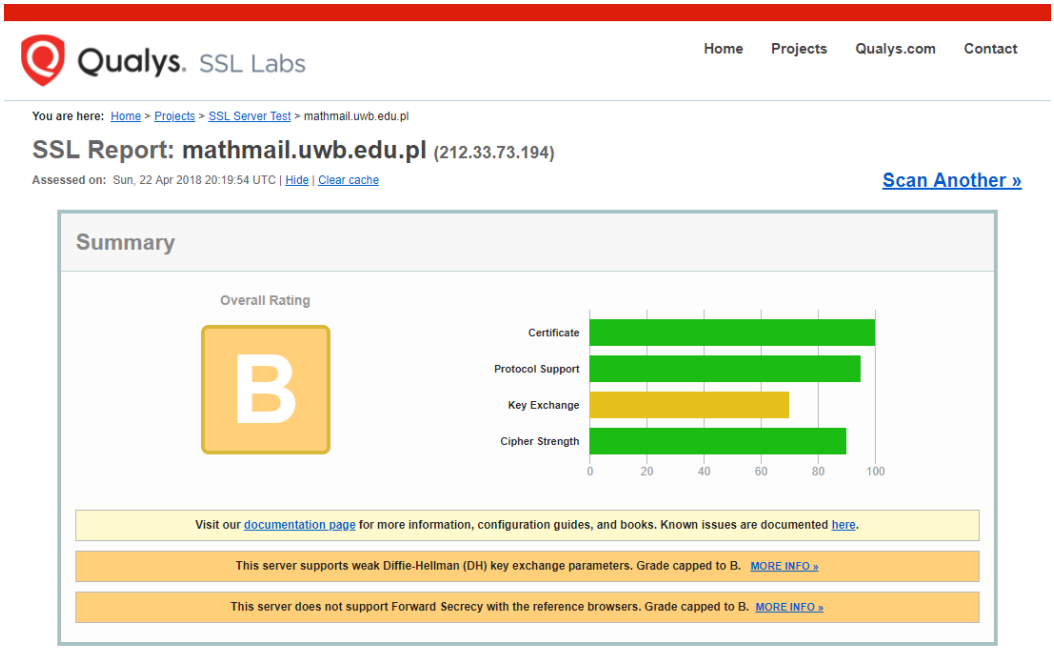
Dwa ostatnie polecenia to sprawdzenie, czy usługa działa poprawnie i jakie procesy w jej ramach zostały uruchomione.

Ponieważ aktualizacja SSH przez SSH jest ryzykowna, warto zostawić sobie otwarte dodatkowe połączenie SSH zanim przystąpi się do pracy.

5.4 Apache 2

Jednym z podstawowych zadań serwera **math** jest hosting strony Wydziału Matematyki i Informatyki `matinf.uwb.edu.pl` oraz Instytutu Matematyki `math.uwb.edu.pl`. Poza tym, pod adresem `mathmail.uwb.edu.pl`, działa tutaj webowy interfejs poczty Horde/IMP używający szyfrowanego protokołu HTTP.

Do tej pory używana była wersja `1.3.42` serwera Apache. Jego ocena na `https://www.ssllabs.com/ssltest/` wypadała relatywnie słabo, a dokładniej, została przyznana ocena **B**. Rysunek 5.1 przedstawia dosyć ogólny rezultat przeprowadzonego testu. Bardziej szczegółowe informacje, zawarte są w sekcji *Dodatki* niniejszej pracy.



Rysunek 5.1: Rezultat testu dla *mathmail.uwb.edu.pl* przed wdrożonymi zmianami

Aby wyeliminować problemy związane z protokołem HTTPS nieodzowna była instalacja Apache 2.

5.4.1 Kompilacja

W czasie kompilacji ostatnia, dostępna wersja to `2.4.29`. Poza różnymi bibliotekami wymaganymi przez serwer Apache, na przykład **OpenSSL** istotnym składnikiem jest system Apache Portable Runtime, w skrócie APR. Jest to zestaw bibliotek zapewniający uniwersalny i jednolity interfejs do wielu komponentów systemu operacyjnego niezależny od platformy. Jest on wymagany

do samej kompilacji serwera Apache i musi być zainstalowany wcześniej. APR jest tak skonstruowany, że nie ma możliwości, aby w jednym systemie koegzystowały ze sobą dwie jego wersje: 64 i 32 bitowa. Jako mniej ważną wersję 32 bitową odłożyliśmy na później.

APR tworzą trzy pakiety, w czasie kompilacji dostępne w wersjach:

- `APR 1.6.3`,
- `APR-util 1.6.1`,
- `APR-iconv 1.2.2`.

Zrezygnowaliśmy z kompilacji `APR-iconv`.

Deweloperzy Apache korzystają ze zmodyfikowanego systemu *GNU build system*. Zamiast opcji w skrypcie `configure` określających lokalizacje poszczególnych składników oprogramowania wydzielono plik `config.layout`, gdzie są one określone. Dla bibliotek APR dodaliśmy swój układ o nazwie `CFW`:

```
<Layout CFW>
    prefix:          /opt/cfw
    exec_prefix:     ${prefix}
    bindir:          ${exec_prefix}/bin
    sbindir:         ${exec_prefix}/sbin
    libdir:          ${exec_prefix}/lib/64
    libexecdir:      ${exec_prefix}/lib/64/apr
    mandir:          ${prefix}/man
    sysconfdir:      ${prefix}/etc
    datadir:         ${prefix}/share/apr
    includedir:      ${prefix}/include/apr
    localstatedir:   /var/cfw/apr
    runtimedir:      ${localstatedir}/run
    installbuilddir: ${localstatedir}/build
</Layout>
```

Po ustaleniu typowego środowiska do kompilacji pierwszego składnika `APR` w wersji 64 bitowej wołamy:

```
./configure \
  --enable-layout=CFW \
  --with-installbuilddir=/var/cfw/apr/build \
  --disable-static
```

Po kompilacji przygotowujemy pakiet `APR-util` wykorzystując ten sam układ w `config.layout`:

```
./configure \
  --enable-layout=CFW \
  --with-apr=/opt/cfw \
  --with-crypto \
```

```
  --with-openssl=/opt/cfw \
  --with-gdbm=/opt/cfw \
  --with-berkeley-db=/opt/cfw \
  --with-mysql=/opt/cfw \
  --with-expat=/usr \
  --with-iconv=/usr
```

Kompilujemy go i przygotowujemy wspólny pakiet instalacyjny zawierający `APR` i `APR-util` tylko w wersji 64 bitowej. Po jego instalacji możemy zacząć kompilację źródeł serwera Apache.

W `config.layout` dla serwera Apache dodajemy nowy układ składników o nazwie `CFW`:

```
<Layout CFW>
    prefix:          /opt/cfw
    exec_prefix:     ${prefix}
    bindir:          ${exec_prefix}/bin
    sbindir:         ${exec_prefix}/sbin
    libexecdir:      ${exec_prefix}/lib/64/httpd
    mandir:          ${prefix}/man
    sysconfdir:      ${prefix}/etc/httpd
    datadir:         ${prefix}/share/httpd
    iconsdir:        ${datadir}/icons
    htdocsdir:       ${datadir}/htdocs
    manualdir:       ${htdocsdir}/manual
    cgidir:          ${datadir}/cgi-bin
    includedir:      ${prefix}/include/httpd
    localstatedir:   /var/cfw/httpd
    runtimedir:      ${localstatedir}/run
    logfiledir:      ${localstatedir}/log
    errordir:        ${localstatedir}/error
    proxycachedir:   ${localstatedir}/proxy
    installbuilddir: ${localstatedir}/build
</Layout>
```

Ilość opcji w skrypcie `configure` serwera Apache jest imponująca. Większość dostępnych funkcjonalności jest na ten moment zbędna dla serwera **math** aczkolwiek warto mieć je dostępne, w celu wykorzystania, w momencie wzrostu wymagań:

```
./configure \
  --enable-layout=CFW \
  --disable-static \
  --enable-mods-shared=all \
  --enable-authn-dbm \
  --enable-authn-anon \
  --enable-authn-dbd \
  --enable-authn-socache \
```

```
--enable-authz-dbm \
--enable-authz-owner \
--enable-authz-dbd \
--enable-auth-form \
--enable-auth-digest \
--enable-allowmethods \
--enable-isapi \
--enable-file-cache \
--enable-cache-disk \
--enable-cache-socache \
--enable-so \
--enable-watchdog \
--enable-macro \
--enable-dbd \
--enable-dumpio \
--enable-echo \
--enable-buffer \
--enable-data \
--enable-ratelimit \
--enable-request \
--enable-include \
--enable-reflector \
--enable-substitute \
--enable-sed \
--enable-deflate \
--enable-xml2enc \
--enable-http \
--enable-log-debug \
--enable-log-forensic \
--enable-logio \
--enable-expires \
--enable-remoteip \
--enable-session \
--enable-session-cookie \
--enable-session-crypto \
--enable-session-dbd \
--enable-ssl \
--enable-privileges \
--enable-info \
--enable-negotiation \
--enable-actions \
--enable-userdir \
--enable-rewrite \
--with-pcre=/opt/cfw \
--with-libxml2=/usr/include/libxml2 \
--with-ssl=/opt/cfw
```

Kompilacja przebiegła w miarę bezproblemowo. Jedyny mankament na jaki natknęliśmy się, to nieprawidłowo ustawione różne warianty `LDFLAGS` w `build/config_vars.mk`. Trzeba je było poprawić w następujący sposób:

```
-L/opt/cfw/lib/64 -R/opt/cfw/lib/64
   -L/usr/sfw/lib/64 -R/usr/sfw/lib/64
```

Niezbędnym oprogramowaniem ściśle związanym z serwerem Apache jest moduł PHP. Zwykle PHP jest instalowane jako oddzielny pakiet. Ponieważ nie zamierzamy używać różnych wersji PHP na jednym serwerze a aktualizacje wykonujemy niezbyt często, wydaje się naturalnym zszycie Apache+PHP w jeden pakiet, tak jak to mam miejsce na przykład w `LAMP` albo `XAMPPP`.

W czasie, gdy kompilowaliśmy PHP dostępna była wersja `5.6.26`.

Przed zawołaniem skryptu `configure` zmieniamy w `CFLAGS` i `CXXFLAGS` opcję `-fast` na `-xO2`. Decyduje ona o poziomie optymalizacji wykonywanych przez kompilator. Opcja `-fast` daje najwyższy możliwy poziom optymalizacji, ale dla kodu PHP musimy go obniżyć. PHP podobnie jak Apache ma sporo różnych funkcjonalności:

```
./configure \
    --prefix=/opt/cfw \
    --with-apxs2=/opt/cfw/bin/apxs \
    --with-imap=/data/devel/build/imap-2007f \
    --with-imap-ssl=/opt/cfw \
    --with-openssl=/opt/cfw \
    --with-libxml-dir=/usr \
    --with-xsl=/usr \
    --with-zlib=/usr \
    --with-bz2 \
    --with-jpeg-dir=/usr \
    --with-png-dir=/opt/cfw \
    --with-gdbm=/opt/cfw \
    --with-db4=/opt/cfw \
    --with-curl=/opt/cfw \
    --with-gettext \
    --enable-soap \
    --with-freetype-dir=/usr/sfw \
    --with-t1lib=/opt/cfw \
    --with-gd \
    --enable-gd-native-ttf \
    --enable-bcmath \
    --enable-calendar \
    --enable-exif \
    --enable-ftp \
    --with-ldap=/opt/cfw \
    --with-gmp=/opt/cfw \
    --with-iconv-dir=/usr \
```

```
--enable-mbstring \
--with-pdo-firebird=/opt/cfw \
--with-pdo-mysql=/opt/cfw \
--with-mysql-sock=/tmp/mysql.sock \
--enable-opcache=no \
--enable-zip \
--with-mysql=/opt/cfw \
--with-mysqli=/opt/cfw/bin/mysql_config \
--with-interbase=/opt/cfw \
--with-mssql=/opt/cfw \
--enable-sockets \
--with-mcrypt=/opt/cfw
```

W trakcie kompilacji objawiło się sporo problemów. Poważniejsze to:

1. do zmiennej `CFLAGS_CLEAN` w głównym `Makefile` trzeba dodać opcję `-I/opt/cfw/include/firebird`,
2. ze zmiennej `EXTRA_LIBS` w głównym `Makefile` trzeba usunąć podłączanie biblioteki `-lltdl`,
3. w `Zend/zend_language_scanner.c` występują DOSowe znaki końca linii `CRLF`, można je usunąć narzędziem `dos2unix`,
4. w plikach `main/php_output.h` oraz `ext/sockets/php_sockets.h` po `PHPAPI ZEND_EXTERN_MODULE_GLOBALS(output);` należy usunąć średnik,
5. w `ext/sockets/conversions.c` oraz `ext/sockets/sendrecvmsg.c` na samym początku należy dodać:

   ```
   #define _XPG4_2
   #define __EXTENSIONS__
   ```

   co włącza odpowiedni standard kompilacji na Solaris dla protokołu SCTP.

5.4.2 Pakiet instalacyjny

Plik `pkginfo` zawierający metadane pakietu:

```
PKG=CFWapache
NAME=Apache - HTTP Server
ARCH=i386
VERSION=2.4.29
CATEGORY=system
DESC=Robust, commercial-grade, featureful HTTP server
VENDOR=http://apache.org/
EMAIL=mariusz@solaris-x86.org
```

```
PSTAMP=Mariusz Zynel, Feb 11, 2018
BASEDIR=/opt/cfw
CLASSES=none manifest
```

Do pakietu dołączony jest manifest usługi `svc:/network/http:apache`. Do naszych potrzeb dostosowujemy konfigurację w `etc/httpd/httpd.conf` oraz w pozostałych plikach w katalogu `etc/httpd/extra`. Włączamy potrzebne moduły i wyłączamy zbędne. Dodajemy moduł PHP:

```
LoadModule php5_module        lib/64/httpd/libphp5.so
```

Modyfikujemy `etc/httpd/extra/httpd-userdir.conf` zmieniając lokalizację katalogów domowych na `/export/home`.

5.4.3 Instalacja

Instalacja nowej wersji Apache i Cyrus IMAP odbyła się 30 kwietnia 2018. Wykorzystaliśmy przeprowadzkę centralnej serwerowni i wyłączenie wszystkich usług w sieci Uniwersyteckiej.

Przed deinstalacją starej wersji robimy kopię wszystkich plików konfiguracyjnych znajdujących się w `/opt/cfw/etc/httpd`. Definicje hostów wirtualnych nie zostaną usunięte podczas instalacji, ale warto zachować pozostałe pliki.

Proces aktualizacji jest standardowy i polega na wyłączeniu usługi, deinstalacji starego pakietu i instalacji nowego:

```
gzcat -s apache-2.4.29-Solaris10-x86.pkg.gz > /tmp/apache
svcadm disable -s apache
pkgrm CFWapache
pkgadd -d /tmp/apache
```

Teraz należy dostosować konfigurację. W `/opt/cfw/etc/httpd/httpd.conf` ustalamy `ServerAdmin` i `ServerName`. W definicji hostów wirtualnych zmieniamy przestarzałe deklaracje:

```
Order allow, deny
Allow from all
```

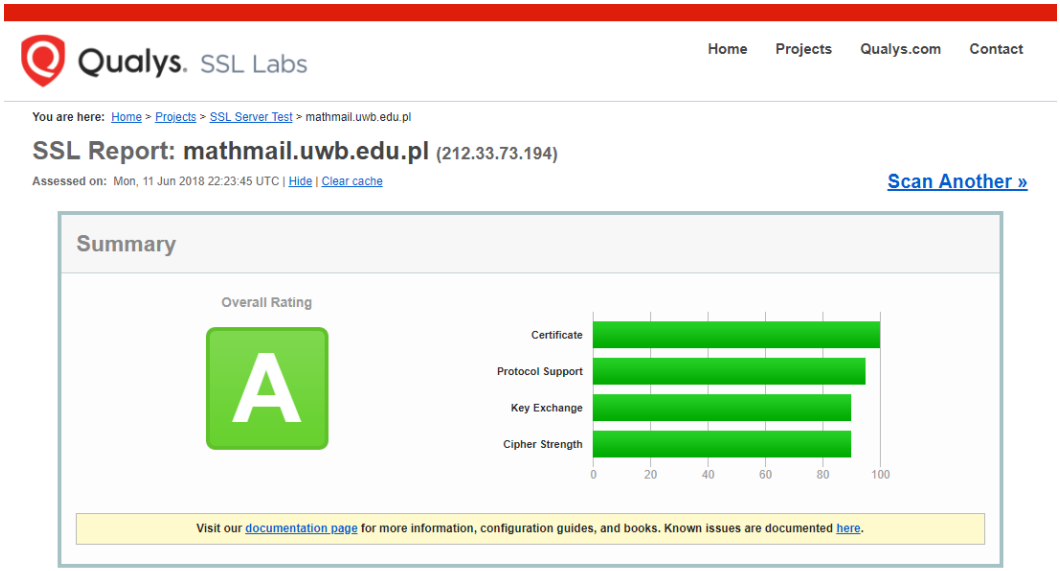
na nowy odpowiednik:

```
Require all granted
```

a w deklaracji `OPTIONS` wszystkie flagi muszą być poprzedzone znakiem + (włącz) lub - (wyłącz). Następnie uruchamiamy usługę, sprawdzamy czy wszystko w porządku i jeśli tak, usuwamy plik tymczasowy:

```
svcadm enable apache
svcs -x
svcs -p apache
rm /tmp/apache
```

Ostateczna ocena testu na stronie `https://www.ssllabs.com/ssltest/`, uzyskana przez serwer po wdrożeniu aktualizacji na serwerze produkcyjnym, to ocena **A**. Ogólny wynik testu przedstawia rysunek 5.2. Bardziej szczegółowe informacje dotyczące rezultatów tego testu, znajdują się w sekcji Dodatki niniejszej pracy.



Rysunek 5.2: Rezultat testu dla *mathmail.uwb.edu.pl* po wdrożonych zmianach

5.5 Cyrus IMAP

5.5.1 Kompilacja

Standardowo przygotowujemy środowisko kompilacji i wykonujemy:

```
PKG_CONFIG_PATH=/opt/cfw/lib/amd64/pkgconfig \
./configure --prefix=/opt/cfw \
  --sbindir=/opt/cfw/sbin \
  --libexecdir=/opt/cfw/sbin \
  --sysconfdir=/opt/cfw/etc/mda \
  --enable-idled \
  --enable-nntp \
  --enable-murder \
  --with-cyrus-user=mail \
  --with-cyrus-group=mail \
  --mandir=/opt/cfw/man \
  --with-mysql=/opt/cfw \
  --with-openssl=/opt/cfw \
  --without-krbdes  \
  --without-snmp \
```

```
  --without-clamav \
  --without-perl
```

W czasie testów, w `imap/mailbox.c` znaleziony został błąd w działaniu jednej z wykorzystywanych funkcji. W systemie Linux, funkcja `printf()` zwraca `(null)`, gdy przekazujemy wartość `NULL` jako wskaźnik na łańcuch do podstawienia `%s`. Solaris pod tym względem zachowuje się konserwatywnie i zgłasza błąd krytyczny dereferencji `NULL`. Błąd znajduje się w programie `reconstruct`, który należy wykonać w trakcie aktualizacji Cyrus IMAP. Oto zapis z core dump:

```
matht# pstack core
core 'core' of 1242:    reconstruct -V max
 ffffffd7ffd3d4b70 strlen () + 40
 ffffffd7ffd43149c printf () + 10c
 ffffffd7fff320269 mailbox_reconstruct_uniqueid () + 81
 ffffffd7fff321784 mailbox_reconstruct () + 160
 000000000040344c ???????? ()
 ffffffd7fff32b737 find_cb () + 22b
 ffffffd7fff111321 myforeach () + 975
 ffffffd7fff32c34e mboxlist_find_category () + 202
 ffffffd7fff32ccf3 mboxlist_do_find () + 987
 ffffffd7fff32cf7d mboxlist_findallmulti () + e9
 ffffffd7fff32cfef mboxlist_findall () + 5f
 0000000000402c22 main () + 696
 00000000004023ab ???????? ()
```

Zastosowana poprawka jest dość oczywista, aczkolwiek znalezienie błędu zajęło sporo czasu:

```
- printf("%s: update uniqueid from header %s => %s\n", mailbox->name,
-       mbentry->uniqueid, mailbox->uniqueid);
+ printf("%s: update uniqueid from header %s => %s\n", mailbox->name,
+       mbentry->uniqueid ? mbentry->uniqueid : "(null)",
+           mailbox->uniqueid);
```

Poza tym:

1. w głównym `Makefile` należy:
   (a) w różnych wariantach `LDFLAGS` poprawić lokalizacje bibliotek 64 bitowych, dopisując `/64`, między innymi `/opt/cfw/lib` zmieniamy na `/opt/cfw/lib/64`,
   (b) do `CFLAGS` należy dodać: `-xc99 -I/usr/include/kerberosv5`,
   (c) do `LIBS` trzeba dopisać `-lkrb5 -lCrun`,
   (d) usunąć wszystkie flagi `-pthread` włączające wątki POSIXowe.
2. wszystkie wystąpienia `LOG_ERR` zamieniamy na `LOG_NOTICE`, co wpływa na sposób umieszczania komunikatów w dzienniku `/var/log/syslog`.

5.5.2 Pakiet instalacyjny

Pakiet oprogramowania Cyrus IMAP nazwaliśmy MDA jako skrót od Mail Delivery Agent bo w końcu zawiera on nie tylko serwer IMAP, ale także POP3.

Plik `pkginfo` zawierający metadane pakietu:

```
PKG=CFWmda
NAME=MDA - Cyrus IMAP/POP3 server
ARCH=i386
VERSION=3.0.5
CATEGORY=application
DESC=Implementation of the Internet Message Access Protocol (IMAP)
  and Post Office Protocol 3 (POP3) with virtual domains support.
WEBSITE=http://www.cyrusimap.org/
EMAIL=mariusz@solaris-x86.org
PSTAMP=Mariusz Zynel, Feb 4, 2018
BASEDIR=/opt/cfw
CLASSES=none manifest
```

Dołączamy manifest usługi `svc:/network/mda` wraz ze związanymi z nim skryptami `i.manifest` oraz `r.manifest`. Aby zautomatyzować dodawanie portów usług IMAP i POP3 do `/etc/services` oraz dopisywanie modułu `Cyrus::IMAP` w Perlu do listy zainstalowanych modułów, napisany został skrypt `postinstall`:

```
# Configure services and Perl modules
#
# Copyright 2018 Marcin Roszkowski & Mariusz Żynel

SERVICESFILE=$PKG_INSTALL_ROOT/etc/inet/services
PERLLOCAL=/opt/cfw/lib/perl5/5.8.8/i86pc-solaris/perllocal.pod

chmod +w $SERVICESFILE
grep "^imaps" $SERVICESFILE >/dev/null ||
    echo "imaps   993/tcp   # Internet Mail Access Protocol over SSL"
        >>$SERVICESFILE
grep "^pop3s" $SERVICESFILE >/dev/null ||
    echo "pop3s   995/tcp   # Post Office Protocol - V3 over SSL"
        >>$SERVICESFILE
grep "^sieve" $SERVICESFILE >/dev/null ||
    echo "sieve   4190/tcp  # Sieve transport service"
        >>$SERVICESFILE
grep "^lmtp"  $SERVICESFILE >/dev/null ||
    echo "lmtp    2003/tcp  # Local Mail Transfer Protocol"
        >>$SERVICESFILE
chmod -w $SERVICESFILE

grep "Cyrus::IMAP" $PERLLOCAL > /dev/null || cat >>$PERLLOCAL <<EOF
```

```
=head2 Wed Mar 7 9:16:18 2018: C<Module> L<Cyrus::IMAP|Cyrus::IMAP>
=over 4
=item *
C<installed into: /opt/cfw/lib/perl5/site_perl/5.8.8>
=item *
C<LINKTYPE: dynamic>
=item *
C<VERSION: 1.00>
=item *
C<EXE_FILES: cyradm>
=back
EOF

exit 0
```

5.5.3 Instalacja i uruchomienie

Przed odinstalowaniem starej wersji pakietu `CFWmda` koniecznie trzeba zrobić kopię baz danych `/var/cfw/mda` oraz konfiguracji `/opt/cfw/etc/mda`. W bazie `mailboxes.db` spisane są wszystkie katalogi użytkowników IMAP/POP3 na **math**, których strata stanowiłaby dość poważny problem. Należy całkowicie wyłączyć pocztę (SMTP oraz IMAP/POP3), czyli usługi `sendmail` oraz `mda`.

Na serwerze **math** wykonujemy aktualizację z wersji `2.4.16` do `3.0.5`. Procedura aktualizacji Cyrus IMAP z wersji 2 do 3 nie jest trywialna. Została ona dokładnie opisana w [10]. Zmianie uległa między innymi wewnętrzna struktura baz danych.

W naszym wypadku procedura aktualizacji wyglądała następująco:

```
gzcat -d mda-3.0.5-Solaris10-x86-sse41.pkg.gz > /tmp/mda
svcadm disable sendmail
svcadm disable mda
gtar cf /backup/tmp/mdaetc.tar -C /opt/cfw/etc/mda .
gtar cf /backup/tmp/mdavar.tar -C /var/cfw/mda .
gtar cf /backup/tmp/datamail.tar -C /data/mail .
pkgrm CFWmda
pkgadd -d /tmp/mda
gtar xf /backup/tmp/mdavar.tar -C /var/cfw/mda
svcadm enable mda
reconstruct -V max
```

Po przetestowaniu IMAP i POP3, program Thunderbird nie zauważył zmiany, co oznaczało pomyślne wykonanie całego procesu. Następnie, można uruchomić SMTP i wykonać niezbędne czynności, w celu usunięcia niepotrzebnych już plików:

```
svcadm enable sendmail
rm /tmp/mda
```

Podsumowanie

Problem utrzymania racjonalnie wysokiego poziomu bezpieczeństwa transmisji danych na serwerach, sprowadza się w zasadzie do regularnych aktualizacji oprogramowania. Z wielu różnych powodów, na serwerze wydziałowym **math** aktualizacje te zostały nieco zaniedbane. Głównym celem mojej pracy była zmiana tej sytuacji. Aby wykorzystać najnowsze algorytmy szyfrowania i przeciwdziałać w ten sposób typowym atakom hakerskim, należało zainstalować najnowsze wersje: biblioteki OpenSSL oraz serwerów SSH, IMAP, POP3 i HTTP. Zadanie to w przypadku Sun Solaris 10 było o tyle trudne, że wymagało uprzedniej kompilacji odpowiedniego oprogramowania. W związku z faktem iż platforma Solaris 10 jest nieco przestarzała i dość niszowa, kompilacja wymagała dostosowania kodu. Na maszynie deweloperskiej często powtarzane były etapy: kompilacja, instalacja, testowanie, poprawki w kodzie i od nowa. Dodatkowo warto także było zadbać o pakiety instalacyjne, aby przygotowane oprogramowanie można było łatwo dystrybuować.

Ostatecznie zaktualizowane zostały wszystkie wymienione wyżej wrażliwe elementy serwera **math** i od dłuższego czasu pracują poprawnie. Na 110 użytkowników serwera math tylko w 3 przypadkach konieczna była pomoc administratora po wykonaniu aktualizacji.

Efekty wykonanej pracy można zweryfikować na SSL Labs (*https:// www.ssllabs.com/*) oraz w przygotowanej dodatkowo na tę okoliczność aplikacji webowej (*http://theta.uwb.edu.pl/ sysadm/*).

Bibliografia

[1] Ivan Ristić, *Bulletproof SSL and TLS*, Feisty Duck, London: 2015

[2] Alexa Internet `https://en.wikipedia.org/wiki/Alexa_Internet`, dostęp: 28.05.2018

[3] Application Packaging Developer's Guide, `https://docs.oracle.com/cd/E26505_01/html/E28550/docinfo.html`, dostęp: 04.02.2018

[4] Asymmetric Encryption `https://www.researchgate.net/figure/Asymmetricencryptionprimitive_fig2_321123382`, dostęp: 26.04.2018

[5] Cipher Block Chaining `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_(CBC)`, dostęp: 27.04.2018

[6] Content Delivery Network `https://en.wikipedia.org/wiki/Content_delivery_network`, dostęp: 27.05.2018

[7] CERT `https://en.wikipedia.org/wiki/Computer_emergency_response_team`, dostęp: 14.05.2018

[8] Clock skew `https://en.wikipedia.org/wiki/Clock_skew`, dostęp: 27.04.2018

[9] Cross-site request forgery `https://en.wikipedia.org/wiki/Crosssite_request_forgery`, dostęp: 05.05.2018

[10] Cyrus IMAP Upgrading to 3.0 `https://cyrusimap.org/imap/download/upgrade.html`, dostęp: 04.02.2018

[11] Deprecating Secure Sockets Layer Version 3.0 `https://tools.ietf.org/html/rfc7568`, dostęp: 11.04.2018

[12] Digital signature `https://searchsecurity.techtarget.com/definition/digitalsignature`, dostęp: 16.04.2018

[13] DNS spoofing `https://en.wikipedia.org/wiki/DNS_spoofing`, dostęp: 09.05.2018

[14] Entropy https://en.wikipedia.org/wiki/Entropy, dostęp: 25.04.2018

[15] Extended Validation Certificates https://www.globalsign.com/en/sslinformationcenter/whatisanextendedvalidationcertificate/, dostęp: 26.04.2018

[16] Initialization vector https://en.wikipedia.org/wiki/Initialization_vector, dostęp: 23.04.2018

[17] Jitter https://en.wikipedia.org/wiki/Jitter, dostęp: 06.05.2018

[18] Malware https://en.wikipedia.org/wiki/Malware, dostęp: 06.05.2018

[19] Online Certificate Status Protocol https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol, dostęp: 03.05.2018

[20] Organization Validation Certificates https://www.entrustdatacard.com/pages/ovvsdv, dostęp: 26.04.2018

[21] Padding (cryptography) https://en.wikipedia.org/wiki/Padding_(cryptography), dostęp: 25.04.2018

[22] Prohibiting Secure Sockets Layer (SSL) Version 2.0 https://tools.ietf.org/html/rfc6176, dostęp: 11.04.2018

[23] Pseudo-Random Functions https://crypto.stanford.edu/pbc/notes/crypto/prf.html, dostęp: 17.04.2018

[24] HTTP Strict Transport Security https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security, dostęp: 16.05.2018

[25] HTTP State Management Mechanism https://tools.ietf.org/html/rfc6265, dostęp: 15.05.2018

[26] HTTP Public Key pinning https://en.wikipedia.org/wiki/HTTP_Public_Key_Pinning, dostęp: 11.05.2018

[27] Transport Layer Security Extensions https://tools.ietf.org/html/rfc3546, dostęp: 11.04.2018

[28] Transport Layer Security (TLS) Renegotiation Indication Extension https://tools.ietf.org/html/rfc5746, dostęp: 19.04.2018

[29] Trusted third party https://en.wikipedia.org/wiki/Trusted_third_party, dostęp: 22.04.2018