



УНИВЕРЗИТЕТ СИНГИДУНУМ
ДЕПАРТМАН ЗА ПОСТДИПЛОМСКЕ СТУДИЈЕ

Безбедносни изазови и механизми заштите код SSL/TLS протокола

- мастер рад -

Ментор:

проф. др Александар Јевремовић

Кандидат:

Милош Милановић

Број индекса: 2016/410240

Београд, 2017.

Садржај

1 Увод.....	3
1.1 Изван криптографије.....	7
1.2 Претње на Интернету.....	8
2 Принцип рада протокола.....	10
2.1 SSL руковање.....	10
2.1.1 Савршена тајност унапред.....	12
2.2 OpenSSL библиотека.....	13
3 Правилно подешавање SSL протокола на Веб серверу.....	16
3.1 Имплементација Let's Encrypt сертификата на Апач Веб сервер.....	16
3.1.1 ACME протокол.....	18
3.2 Правилан избор криптографских алгоритама.....	18
3.3 Корисна заглавља протокола за пренос хипертекста.....	19
3.3.1 HPKP механизам.....	20
3.3.2 HSTS механизам.....	21
3.3.3 Остала корисна заглавља и OWASP препоруке.....	22
3.4 Механизми за информисање о статусу ревокације код X509 сертификата.....	23
3.5 Заштита приватних кључева.....	25
3.6 Остали методи заштите.....	26
4 Напади на SSL протокол.....	28
4.1 Heartbleed напад.....	28
4.2 Како Heartbleed ради.....	29
4.2.1 Ишчитавање садржаја радне меморије.....	31
4.3 Padding oracle напад.....	34
4.3.1 Објашњење напада, CBC мод и криптографски оракл.....	34
4.3.2 Варијације на Padding oracle напад.....	36
4.4 Напади преко бочних канала - проблеми са компресијом.....	36
4.4.1 Алгоритам компресије.....	37
4.4.2 CRIME напад.....	38
4.4.3 BREACH напад.....	39
4.4.4 Временски напади - TIME и HEIST.....	39
4.5 Криптографски пропусти.....	40
4.5.1 FREAK напад.....	40
4.5.2 Logjam напад.....	42
5 Напад "Човек у средини".....	44
5.1 Подешавање HTTPS проксија на Burp Suite алату.....	45
5.2 Тестирање Веб апликација користећи Burp.....	45
5.3 Тестирање мобилних апликација.....	47
6 Тестирање конфигурације SSL протокола.....	50
6.1 API за тестирање.....	54
7 Закључак.....	56

1 Увод

Култни филм "Блејд ранер" Ридлија Скота из 1982. године предвиђао је да ће до 2019. године доћи до колонизације свемира, хуманоидних робота и летећих аутомобила. Иако се те ствари за сада нису оствариле, успон технологије на самом почетку 21. века наговештава правац у којем се цивилизација креће. Ипак, утисак је да се о свим тим футуристичким концептима још не би водила дискусија да '70-их година прошлог века није дошло до изума Интернета, који је омогућио људима ствари које су биле незамисливе у релативно блиској прошлости.

Интернет мрежа представља јединствен комуникациони систем у историји људске цивилизације[1]. Сервиси који користе Интернет као инфраструктуру као што су Веб сервис, сервис електронске поште, VoIP и сл. су постали тешко заменљив део свакодневнице модерног живота. Могуће је управљати рачуном у банци путем мобилног телефона, учествовати у видео позиву са особом на другом крају света са неосетним кашњењем у комуникацији, школовати се или радити у иностранству а не напуштати кућу. Астронаути у међународној свемирској станици (енгл. *International Space Station - ISS*) такође могу комуницирати са својим најближима путем Интернета¹. Једно занимљиво истраживање које је објавио доктор Мајкл Косински (енгл. *Michael Kosinski*) демонстрира алгоритам предиктивне анализе над стварима које је особа лајковала на Фејсбуку, и у зависности од броја лајкова може са великом прецизношћу погодити пол, етничку припадност, сексуалну оријентацију, религиозне и политичке погледе, личне особине, брачни статус родитеља и сличне ствари[2] - ово говори о моћи једног наизглед безазленог концепта који користи Интернет и његов Веб сервис као инфраструктуру.

Док постоје многи заговорници тезе како је Интернет као технологија имао огроман утицај на отуђивање међу људима и уназадио човека, чињеница је да, као и сви други изуми уосталом, Интернет са собом носи и позитивне и негативне стране. Још од зоре човечанства, човек је показивао одређену интелектуалну надмоћ над осталим облицима живота. Да би се у почетку одбранио од великих животиња, човек је изумео оружје. У почетку је то оружје било једноставно, нпр. обично дрво заострено на једном крају (копље). Са еволуцијом човека, дошло је и до еволуције оружја јер човек има потребу за тим, и данашње оружје представља нормалну еволуцијску промену у историји човечанства. То оружје може бити коришћено за одбрану од великих животиња, што је генерално добра ствар за човека, али може бити коришћено и за лоше ствари, нпр. од стране терористичких екстремиста или криминалаца. Исто као што је човек измислио оружје јер је имао потребу за самоодбраном, измишљен је и Интернет, јер човек има потребу за комуникацијом. Исто као и оружје, Интернет може бити коришћен у добре сврхе - комуникација, едукација, дружење; док исто тако може бити коришћен и за лоше ствари, нпр. сајбер криминал.

1 <https://twitter.com/stationcdrkelly/status/592233875063107584>

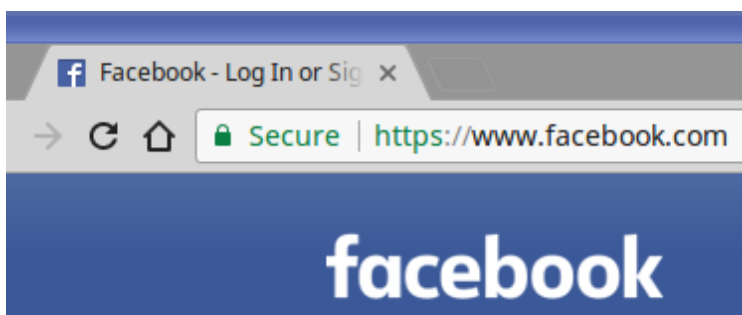
Узевши у обзир број корисника који се ослања на њих, сервиси који користе Интернет као инфраструктуру морају да обезбеде и одређен ниво заштите. Срећом, због модуларне архитектуре Интернета (OSI слојеви) могуће је лако имати протокол који може бити коришћен за заштиту више различитих сервиса.

#	Слој	Протоколи
7	Слој апликације	HTTP, FTP, POP, IMAP, SMTP, SSH, NTP, DNS, DHCP
6	Слој презентације	
5	Слој сесије	TLS/SSL
4	Транспортни слој	TCP, UDP
3	Мрежни слој	IP, ICMP, ARP
2	Слој везе података	802.3, 802.11, Bluetooth
1	Физички слој	-

Табела 1. OSI слој

Напомена: OSI модел по својој природи не представља научну дефиницију, већ апстрактни модел архитектуре мреже. Подељена су мишљења на ком слоју TLS и SSL протоколи раде, али најчешће се узима да је то пети слој, пошто свакако мора бити изнад транспортног слоја (користи TCP протокол).

TLS (енгл. *Transprot Layer Security*), као и његов претходник SSL (енгл. *Secure Sockets Layer*) представљају криптографске протоколе који нуде заштиту у комуникацији путем Интернета, који сам по себи не представља безбедан комуникациони канал. Сам назив TLS никад није заживео, па се често и у литератури под називом SSL мисли на исти протокол. Протокол се може користити за заштиту више L7 сервиса - Веб, е-пошта, FTP протокол итд. На Вебу TLS протокол се користи да обезбеди заштићену комуникацију између Веб сервера и клијента, користећи безбедну верзију протокола за пренос хипертекста (енгл. *HyperText Transfer Protocol Secure* - *HTTPS*).

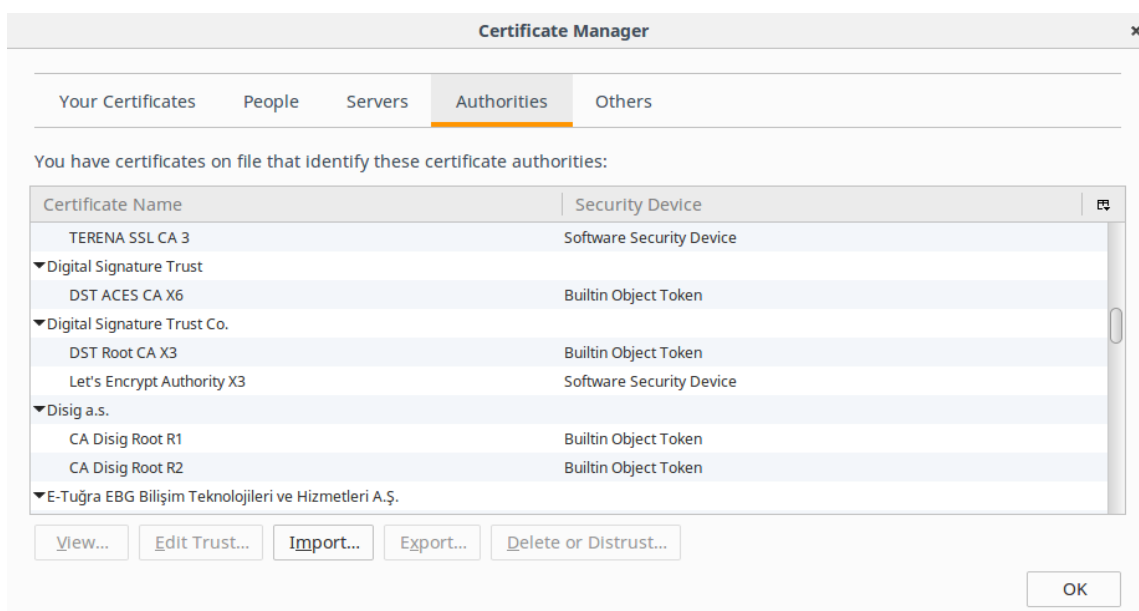


Слика 1. Верификација сертификата Веб сервера код Гугловог Chrome прегледача

SSL протокол решава питање поверљивости и интегритета података у комуникацији на Интернету. Поверљивост у комуникацији се постиже користећи симетрично шифровање над подацима који се преносе. Кључ се може генерисати договарањем (нпр. Дифи-Хелман протокол за размену кључа) или може бити генерисан на једној страни (сервера), шифрован

приватним кључем те стране и прослеђен другој страни. Интегритет у комуникацији користећи једносмерно шифровање, тј. хеш функције. Ово се постиже тако што се на једној страни рачуна хеш вредност поруке. Та вредност се шаље уз поруку и рачуна опет на другој страни. Ако се те две вредности поклапају, можемо закључити да је интегритет података сачуван.

На Вебу се клијент серверу аутентификује користећи аутентификацију на нивоу директоријума (*Basic* и *Digest* аутентификација) или на нивоу апликације (систем за пријављивање у оквиру саме апликације)[1]. Пошто се идентификатор сесије који се користи за даљу ауторизацију при раду клијента прослеђује у сваком наредном захтеву због природе HTTP протокола[3], јасно је да је клијентска страна аутентификована. SSL протокол нуди могућност да се сервер "аутентификује" клијенту, користећи PKI (енгл. *Public key infrastructure*) Веб сервиса. Ово се постиже уколико клијент верује трећој страни од поверења - сертификационом телу (енгл. *Certificate authority* - *CA*). Потребно је да клијентов Веб читач верује овом сертификационом телу (CA се може наћи у предефинисаној листи у оквиру читача или може бити додат ручно од стране клијента у њу), односно да је серверов сертификат који је прослеђен клијенту потписан од стране тог сертификационог тела. На слици испод приказана је компонента за менаџмент трећих страна од поверења у оквиру Мозила Фајрфокс прегледача. Уколико је сертификат нашег Веб сервера издат од стране сертификационог тела које се не налази у овој листи код клијента, тада ће клијент бити упозорен од стране Веб прегледача пре него што се захтевани ресурс са нашег сајта прикаже.



Слика 2. *Certificate Manager* код Mozilla Firefox прегледача

Дакле, сертификационо тело је ентитет коме клијент и сервер верују. Посао сертификационог тела је да изда дигитални сертификат серверу који потврђује дигитални идентитет сервера на мрежи. Сертификат треба да садржи јавни кључ сервера и датум истека валидације сертификата, а опционо може садржати име власника и сл. Сервер да би добио сертификат мора да пошаље захтев за потписом сертификата (енгл. *Certificate signing request*

- CSR). Ова услуга се наплаћује, мада новији пројекат под називом Let's Encrypt CA представља сертификационо тело које ову услугу нуди бесплатно.

Сертификациона тела функционишу по нивоима. Сертификациона тела првог нивоа се називају корена сертификациона тела (енгл. *Root CA*). Ово је слично као код архитектуре Интернет мреже и провајдера. Сертификационо тело нижег нивоа треба бити верификовано код сертификационог тела на вишем нивоу итд. (као што рутери корисника морају бити повезани на модеме Интернет провајдера најнижег нивоа, који опет имају свог Интернет провајдера итд.). Сертификат се може повући ручно - ревокација (енгл. *revocation*) или може истећи (енгл. *expire*), што је логично, чешћи случај.

Постоје и самопотписани сертификати - ово се односи на сертификате који су потписани својим приватним кључем. Као и сертификати потписани од неадекватног сертификационог тела ови сертификати се не требају користити у комерцијалне сврхе (неће бити валидни). Ипак, треба имати у виду да могу бити корисни, нарочито за заштиту од пасивних напада у оквиру локалних мрежа (нпр. прислушкивање) или за потребе тестирања у оквиру комплекснијих развојних окружења.

Због тога што је коришћење асиметричне криптографије најчешће подразумевало коришћење RSA алгоритма, компанија коју држе аутори тог алгоритма, RSA Security, Inc., је у циљу промоције криптографије '90-их година кренула са објављивањем стандарда под називом PKCS (енгл. *Public Key Cryptography Standards*). Иако нису представљали индустријске спецификације, већ практична упуства, PKCS стандарди су постали *de facto* стандарди индустрије и у последњих пар година разматра се чак и њихова званична стандардизација од стране IETF (енгл. *Internet Engineering Task Force*) организације. Значајни су за ову причу јер дефинишу стандарде за CSR (PKCS #10). Иако је у оквиру PKCS дефинисан и стандард за складиштење приватног кључа заједно са сертификатом (PKCS #12), у пракси се чешће среће употреба PEM (енгл. *Privacy Enhancement Message*) формата (IETF стандард).

Највећа ствар коју једна Интернет компанија у Америци данас може да учини без саветовања са адвокатима како би заштитила права корисника широм света је да омогући SSL шифровање на свакој Веб страници коју посетите. Ово је данас битно зато што ако погледате примерак неке књиге на Амазону, NSA може то да види, као и руска обавештајна служба, кинеска обавештајна служба... Сви могу да виде то јер нема шифровања. Светска библиотека је на Амазону, а они немају шифровање. Ово морамо да променимо - не само за Амазон, не мислим да их издвајам, али су одличан пример. Све компаније морају да пређу на подразумевано SSL шифровање за све кориснике. То ће повећати приватност и права која ће људи уживати широм света.

Едвард Сноуден (Edward Snowden), "Како вратити Интернет?", TED конференција, 2014. година

Иако постоје и друга решења, за потребе генерисања кључева се најчешће користи OpenSSL библиотека - пројекат отвореног кода написан у С језику, доступан на Github репозиторијуму². Једна од највећих новчаних награда на HackerOne платформи која плаћа хакере широм света за откривање пропуста у популарним Веб апликацијама / сервисима је била исплаћена управо за откривање пропуста у овој библиотеци.

Актуелна верзија протокола је TLS 1.2, док је TLS 1.3 у фази драфта. Због сигурносних пропуста у првој верзији SSL 1.0 она никада није званично објављена. Верзије SSL 2.0 и 3.0 су депрактиковане и воде се као небезбедне, док безбедност код верзија TLS 1.0, 1.1 и 1.2 није подразумевана, већ зависи од конфигурације. О томе, као и о нападима на SSL протокол, више у наставку рада.

Верзија	Год.
SSL 1.0	није објављен
SSL 2.0	1995
SSL 3.0	1996
TLS 1.0	1999
TLS 1.1	2006
TLS 1.2	2008
TLS 1.3	драфт

Табела 2. Верзије SSL/TLS протокола

1.1 Изван криптографије

Безбедност и заштита на Интернету данас представља све значајније и све популарније питање[4]. SSL је уз виртуалне приватне мреже (енгл. *Virtual Private Networks - VPN*) један од главних стубова Интернет анонимности и пројеката као што је Tor. Док се SSL користи за заштиту комуникације, VPN је коришћена за заштиту извора. Сам SSL је значајан за свакодневне ствари попут пријављивања на налог на друштвеној мрежи, онлајн плаћање, слање интерних поверљивих података компаније, али у земљама у којима влада диктатура, а новинарима није обезбеђена слобода говора, Интернет као медијум за објављивање вести и овакви концепти заштите добијају потпуно другачији контекст. То је јасно када погледамо колико ресурса NSA улаже у програме као што је BULLRUN, или још драстичнији пример кинеске владе и чувеног Кинеског заштитног зида[5]. SSL као што је напоменуто сам по себи не игра главну улогу у концептима као што су Интернет анонимност и Веб поверења (енгл. *Web of trust*), али је јако битан у техничком аспекту целе те приче. Информација је моћ, и у случају државних агенција које шпијунирају своје грађане, информација им даје моћ над њима (подсетимо да су Сједињене Америчке Државе некада контролисале извоз

² <https://github.com/openssl/openssl>

криптолошких алгоритама са истим третманом као да се ради о оружју[6]). Ту су и глобални примери, попут Гугла или Фејсбука. Познати амерички хакер Џејкоб Еплбаум (енгл. *Jacob Appelbaum*) је једном изјавио како никаква количина насиља неће решити математички проблем, тако да криптографски концепти могу решити многе политичке проблеме данашњице.

1.2 Претње на Интернету

Џулијан Асанж (енгл. *Julian Assange*), оснивач Викиликса, један је од најгласнијих у тврдњама да иза Интернет гиганта попут Гугла, Амазона, Фејсбука итд. стоје државне службе. Ово значи да сва та количина података које остављамо на Интернету, може бити доступна и државним службама. Овде имамо два супротна погледа:

- државне службе требају имати приступ тој количини података јер може помоћи у спречавању терористичких актова
- државне службе не требају имати приступ тим подацима јер корисници на Интернету желе одређени ниво приватности и анонимности

Сајбер криминал постаје једна од водећих грана криминала - Wannacry, Petya и ZCrypt рансомвери (енгл. *ransomware*) су нанели огромну финансијску штету, као и немерљиву штету по репутацију многих компанија током претходне (2017) године. Осетан је бијас који многи људи имају према сајбер криминалу, где се потцењивачки гледа на сајбер нападе, што још више улива храброст криминалцима да се окушају у овим водама. Према неким статистикама највише испаштају здравствене институције[7], оно што је и најбитније обичном човеку, а одмах за њима следе владине организације и компаније које се баве малопродајом. Новац је све више и више на Интернету, а трендови као што су раст крипто валута то и потврђују.

У последње време постају све више популарни отворени bug bounty програми - где компанија позива безбедносне стручњаке и ентузијасте да када открију пропуст у Веб или мобилној апликацији компаније пријаве то компанији и добију адекватну награду. Проблем је што су нападачи почели да плаћају више од самих компанија - менаџмент тимови унутар компанија не вреднују безбедносне пропусте кроз штету колику могу направити. Такође, није ни редак случај да особа која пријави пропуст ни не добије обећану награду³. Компаније често балансирају са улагањем у заштиту од сајбер криминала, гледајући тзв. ROI параметар (енгл. *Return on investment*). Многи стручњаци за безбедност у сајбер простору попут Бруса Шнајера (енгл. *Bruce Schneier*) и Илија Колоченко (енгл. *Ilija Kolochenko*) су писали о овоме[8] [9], и слажу се да безбедност није улагање које пружа могућност да се заради више у случају добро одрађеног посла, већ трошак, који треба да отплати себе временом. Безбедност је више

3 <https://www.entrepreneur.com/article/227947>

до превенције губитака, него до финансијског добитка компаније. Наравно, из угла компаније, смањивање трошкова је исто што и прављење профита за њихов буџет.

Једна од три највеће кредитне агенције у историји, амерички кредитни биро *Equifax*, овог лета је доживео један од највећих хакерских напада у историји Веба, где су хакери дошли до личних података (име и презиме, датум рођења, адресу, број социјалног осигурања, број возачке дозволе и сл.) више од 143 милиона америчких грађана, што је скоро половина популације САД. Напад није озбиљан само због украдених података, већ и због контроверзног начина на који је компанија одговорила. Информација о нападу је прикривана два месеца док је *Bloomberg News* није пренео први, што је потегло питање да ли је компанија додатно угрозила безбедност својих клијената. Са великом моћи долазе и велике обавезе, и корисници Интернета генерално немају поверења у компаније и државне службе да могу те обавезе да испуне.

2 Принцип рада протокола

SSL протокол на Вебу се користи за заштиту приватности и аутентификацију Веб сервера. Додатно, многе напредне функционалности језика за означавање верзије хипертекста 5 верзије (енгл. *HyperText Markup Language 5 - HTML5*) као што су геолокација и видео и аудио подршка су доступне једино ако се користи HTTPS.

SSL комуникација између клијента и сервера почиње фазом руковања (као код TCP протокола)[10]. У случају да се користи PFS, кључ се генерише за сваку нову сесију. Треба имати у виду да SSL протокол аутентификацију заснива на ономе што корисник зна - тиме протокол прави баланс између сигурности и заштите, занемаривајући правила безбедности и аутентификације за нечим што корисник има и нечим што корисник јесте[11].

2.1 SSL руковање

Комуникација преко HTTPS протокола се одвија након фазе иницијализације која се назива SSL руковање (енгл. *SSL handshake*). По корацима би то изгледало отприлике овако:

1. Клијент шаље тзв. *Client Hello* поруку:
 - подржана верзија - TLS 1.2
 - подржани пакети шифара (енгл. *Cipher suite*) на страни клијента -
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256,
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_128_GCM_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_3DES_EDE_CBC_SHA
 - метод компресије - опционо поље
 - низ случајних бајтова - касније коришћен за генерисање симетричног кључа
2. Сервер одговара са *Server Hello* поруком:
 - изабране шифре - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

- избрани метод компресије
 - идентификатор SSL сесије
 - низ случајних бајтова - касније коришћен за генерисање симетричног кључа
3. Сервер шаље клијенту свој сертификат
 4. Сервер шаље клијенту ECDHE параметре за генерисање симетричног кључа
 - *Server Hello Done*
 5. Клијент шаље своје ECDHE параметре (претпостављамо да је сертификат верификован успешно) и шаље прву шифровану поруку (*Client Hello Request*)
 6. Сервер шаље прву шифровану поруку (*Server Hello Request*)

У наставку комуникације сав саобраћај се шифрује симетричним кључем који је договорен помоћу ECDHE алгорита у овом случају. Симетрично шифровање се користи због брзине у односу на асиметрично шифровање, које је због целе PKI инфраструктуре сигурније у односу на симетрично.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.037131810	192.168.0.104	dualstack.i.ssl.global.fastly.net	TCP	74	40452 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PE...
6	0.065489793	dualstack.i.ssl.global.fastly.net	192.168.0.104	TCP	74	443 → 40452 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1...
7	0.065522460	192.168.0.104	dualstack.i.ssl.global.fastly.net	TCP	66	40452 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=9360...
8	0.065734152	192.168.0.104	dualstack.i.ssl.global.fastly.net	TLSv1.2	251	Client Hello
9	0.094779649	dualstack.i.ssl.global.fastly.net	192.168.0.104	TLSv1.2	1514	Server Hello
10	0.094806681	192.168.0.104	dualstack.i.ssl.global.fastly.net	TCP	66	40452 → 443 [ACK] Seq=186 Ack=1449 Win=32128 Len=0 TSval...
11	0.094827246	dualstack.i.ssl.global.fastly.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
12	0.094832268	192.168.0.104	dualstack.i.ssl.global.fastly.net	TCP	66	40452 → 443 [ACK] Seq=186 Ack=2897 Win=35072 Len=0 TSval...
13	0.095018941	dualstack.i.ssl.global.fastly.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
14	0.095028704	192.168.0.104	dualstack.i.ssl.global.fastly.net	TCP	66	40452 → 443 [ACK] Seq=186 Ack=4345 Win=37888 Len=0 TSval...
15	0.095171880	dualstack.i.ssl.global.fastly.net	192.168.0.104	TLSv1.2	2770	Certificate, Certificate Status, Server Key Exchange, Se...
16	0.095181197	192.168.0.104	dualstack.i.ssl.global.fastly.net	TCP	66	40452 → 443 [ACK] Seq=186 Ack=7049 Win=43392 Len=0 TSval...
17	0.098261135	192.168.0.104	dualstack.i.ssl.global.fastly.net	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Hello Request, ...
18	0.098331182	192.168.0.104	dualstack.i.ssl.global.fastly.net	TLSv1.2	415	Application Data
19	0.121928115	dualstack.i.ssl.global.fastly.net	192.168.0.104	TLSv1.2	308	New Session Ticket, Change Cipher Spec, Encrypted Handsh...

▶ Frame 8: 251 bytes on wire (2008 bits), 251 bytes captured (2008 bits) on interface 0
 ▶ Ethernet II, Src: CompalIn e6:4c:96 (1c:39:47:e6:4c:96), Dst: Tp-LinkT b7:ad:ce (30:b5:c2:b7:ad:ce)
 ▶ Internet Protocol Version 4, Src: 192.168.0.104 (192.168.0.104), Dst: dualstack.i.ssl.global.fastly.net (151.101.0.67)
 ▶ Transmission Control Protocol, Src Port: 40452, Dst Port: 443, Seq: 1, Ack: 1, Len: 185
 ▼ Secure Sockets Layer
 ▶ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
 Content Type: Handshake (22)
 Version: TLS 1.0 (0x0301)
 Length: 180
 ▶ Handshake Protocol: Client Hello

Слика 3. Снимак SSL саобраћаја коришћењем Wireshark алата

На слици изнад приказан је део мрежног саобраћаја који укључује фазу SSL руковања. Пошто се SSL пакет енкапсулира унутар TCP пакета, пре SSL руковања мора да дође до TCP руковања (енгл. *TCP 3-Way Handshake*). У отвореном тексту су прослеђене избрани крипто параметри и јавни кључ сервера, док је остатак саобраћаја шифрован. Осим крипто параметара и сертификата сервера, за корисника је битно да има у виду да ће још неки делови бити прослеђени у отвореном тексту, пре него што SSL сесија почне:

- Адреса сервера
- Упити DNS (енгл. *Domain Name System*) серверу

Такође, серверов сертификат ће садржати име домена везано за сертификат (што не мора у сваком случају бити домен Веб сајта којем приступамо).

Због тога што су операције асиметричне криптографије (дигитални потпис, шифровање јавним кључем, дешифровање приватним кључем) временски захтевне, да би се избегла њихова стална употреба током SSL руковања, SSL нуди могућност да се ово избегне тако што ће сервер послати клијенту идентификатор сесије (Session ID) који ће клијент повезати са тим сервером. Након тога, уместо стандардног процеса SSL руковања потребно је само да клијент и сервер деле исту главну тајну и руковање ће бити успешно). Пошто су у оквиру *Server Hello*, односно *Client Hello*, размењени случајни параметри, генерисани кључеви за нову конекцију ће се разликовати у односу на прошлу.

2.1.1 Савршена тајност унапред

Савршена тајност унапред (енгл. *Perfect Forward Secrecy* - *PFS*) представља метод у криптографији где безбедног сесијског кључа не зависи од безбедности јавног кључа. Значајан је за SSL протокол јер у случају компромитовања сертификата Веб сервера, раније снимљена (шифрована) сесија између клијента и сервера не може бити дешифрована.

Пре имплементације савршене тајности унапред код HTTPS протокола постојао је проблем где је нападачу у случају да поседује шифроване поруке између клијента и сервера потребан само приватни кључ сервера да би их дешифровао. Ово је било могуће јер је коришћена тајна - *Pre-Master Secret*, била генерисана на случајан начин на страни клијента, а затим шифрована јавним кључем сервера. Тајна служи за генерисање сесијског кључа који ће бити коришћен - *Master Secret*. Тада страна која поседује одговарајући приватни кључ (Веб сервер) може једина да дешифрује вредност тајне. У случају да нападач дође до приватног кључа сервера на било који начин, он је у стању да дешифрује све поруке.

Након имплементације савршене тајности унапред ово више неће бити могуће. PFS функционише тако што је размена кључа ефемерна, тј. краткотрајна. Прецизније, за сваку сесију између клијента и сервера биће генерисан нови симетрични кључ. Једини захтев за ово је да се изабере пакет алгоритама који користи DH алгоритам за размену кључа - код TLS протокола последње верзије то су DHE-RSA, DHE-DSA, ECDHE-RSA, ECDHE-ECDSA. Пошто се симетрични кључ се генерише на основу DH протокола, тј. није шифрован јавним кључем Веб сервера, ништа се не штити приватним кључем сервера. То значи да у случају да нападач дође до приватног кључа Веб сервера, он неће моћи преко њега да добије сесијски кључ. А чак и ако нападач дође у посед сесијског кључа, он ће важити само за одређену сесију - врло краткотрајно и тешко може бити од некакве користи нападачу.

Кораци код Дифи-Хелман алгоритма:

1. Договоре се јавне вредности, прост број p и основа g
2. Алиса генерише тајну вредност a и шаље Бобу $A = g^a \bmod p$
3. Боб бира тајну вредност b и шаље Алиси $B = g^b \bmod p$

4. Алиса рачуна $Ks = B^a \bmod p$
5. Боб рачуна $Ks = A^b \bmod p$
6. Алиса и Боб сад деле заједничку тајну - Ks

2.2 OpenSSL библиотека

OpenSSL је библиотека отвореног кода која нуди имплементацију SSL/TLS протокола, уз основне криптографске и помоћне функције. Изворни кôд је доступан јавно, библиотека је написана у С језику, али је портована на остале језике па је на тај начин олакшано коришћење у програмском језику нижег нивоа као што је нпр. PHP, програмски језик који се најчешће користи за програмирање на страни Веб сервера. Актуелна верзија је 1.1.0 (1.1.1 још у фази драфта). Популарни форкови су LibreSSL (коришћен код OpenBSD система) и, занимљиво, Гуглов форк BoringSSL.

Користећи OpenSSL можемо једноставно путем командне линије генерисати пар кључева и захтев за издавањем сертификата, који у суштини представља јавни кључ и мета-подаци о власнику сертификата. Тај захтев затим шаљемо одговарајућем сертификационом телу које га потписује, и тада имамо валидан сертификат за коришћење на Вебу. Такође, путем OpenSSL програма можемо потписати тај кључ самим собом (енгл. *self-signed certificate*). У том случају сертификат неће бити валидан на Вебу (осим ако нисмо валидно сертификационо тело). У случају да желимо да користимо RSA пар кључ величине 2048 бита и сачувамо га у фајлу *domain.key* проследићемо *genrsa* аргумент *openssl* програму, а затим као аргументе *genrsa* функцији проследити даље параметре:

```
openssl genrsa -out domain.key 2048
```

Када генеришемо захтев за потписивањем сертификата користимо *req* компоненту:

```
openssl req -new -sha256 -key domain.key -out domain.csr
```

За разлику од претходне команде која ће сама генерисати пар кључева (у случају да нисмо проследили и *-passout* аргумент), ова команда отвара интерактиван конзолни дијалог где морамо дати вредности о власнику домена за следећа поља:

- *Common Name* - пуно име домена нашег Веб сајта, нпр. www.example.com. Могуће је користити исти сертификат и на под-доменима, у том случају прослеђујемо цокер (енгл. *wildcard*) оператор - *.example.com. Ово поље је једино оавезно поље где морамо унети неку вредност у овом промπτу.

- *Organization Name* - пун назив фирме или организације. Ово укључује и одговарајући суфикс попут Ltd., Inc., и слично.
- *Organizational Unit Name* - департман (нпр. HR, IT...)
- *Country Name* - ISO код државе (нпр. US, RS...)
- *State or Province Name* - име државе, или провинције у случају САД нпр.
- *Locality Name* - име града
- *Email Address* - адреса електронске поште

Верификујемо CSR користећи исту функцију, само другачије аргументе функције:

```
openssl req -noout -text -in domain.csr
```

Након тога у случају да је све у реду шаљемо захтев у сертификационо тело по избору. Популарна сертификациона тела су Verisign, RapidSSL и LetsEncrypt.

Кључеви представљају дуже низове бајтова, и као такви нису претерано људски читљиви. За идентификовање кључа се користи његов отисак (енгл. *fingerprint*, код Мајкрософт система *thumbprint*). У основи, то је само хеш вредност тог низа. Уколико желимо да погледамо отисак кључа користећи OpenSSL програм, користимо команду:

```
openssl x509 -in domain.cer -noout -sha256 -fingerprint
```

Још једна корисна команда, која се углавном користи као алат за дијагностику при конфигурацији SSL протокола на неком сервису, је `s_client`. Уколико желимо да сачувамо одговор сервера у фајлу `debug.txt` можемо користити команду (пошто наредба покреће интерактиван дијалог, пајпована је `echo` команда како би процес био аутоматски затворен):

```
echo "Q" | openssl s_client -connect example.com:443 > debug.txt
less debug.txt
```

`s_client` такође нуди додатне опције које могу проверити да ли сервер подржава одређену верзију SSL/TLS протокола. Још један алат који је користан за дебаговање SSL протокола је Nmap. Иако је превасходно намењен за локалну мрежу и нападе извиђања, једна од корисних додатних функционалности Nmap алата је његов скриптинг енџин (енгл. *Nmap Scripting Engine* - *NSE*). Скрипте се пишу у Lua програмском језику и већ Nmap инсталиран стандардно долази са великим бројем корисних скрипти. Ако желимо да тестирамо које су подржани алгоритми за шифровање на серверу можемо користити преинсталирану скрипту `ssl-enum-ciphers`. Наредба је следећа:

```
nmap --script ssl-enum-ciphers example.com
```

Испод су дати неки примери за још неке операције које OpenSSL подржава.

```
# Хеш функције
echo -n 'Hello!' | openssl dgst -md5
echo -n 'Hello!' | openssl dgst -sha256

# Base64 кодовање и декодовање
echo -n 'Hello!' | openssl enc -base64
echo 'SGVsbG8h' | openssl enc -base64 -d

# Генератор псеудо-случајних бројева
openssl rand -hex 16

# Симетрично шифровање
echo 'Hello!' > plain.txt
openssl rand -out key.bin 256 # Генерисање кључа
openssl enc -e -aes-256-cbc -base64 -in plain.txt -out cipher_text.txt
-pass file:key.bin

# Симетрично дешифровање
openssl enc -d -base64 -aes-256-cbc -in cipher_text.txt -out decrypted.txt
-pass file:key.bin
md5sum plain.txt decrypted.txt # Провера

# Генерисање пара RSA кључева (у шифрованом формату)
openssl genrsa -des3 -out private.pem 2048

# Екстрактовање јавног RSA кључа
openssl rsa -in private.pem -outform PEM -pubout -out public.pem

# Шифровање поруке јавним кључем
openssl rsautl -encrypt -inkey public.pem -pubin -in plain.txt -out
encrypted.bin

# Дешифровање поруке приватним кључем
openssl rsautl -decrypt -inkey private.pem -in encrypted.bin
```

3 Правилно подешавање SSL протокола на Веб серверу

Због модуларности у архитектури рада Интернет мреже SSL протокол се може користити и за заштиту других сервиса - електронске поште, FTP, XMPP и сл. Ипак, Веб сервис је убедљиво најпопуларнији сервис Интернета, тако да је у овом поглављу обрађена правилна имплементација SSL протокола на Веб сервер. Како је Апач најпопуларнији сервер отвореног кода, изабран је за потребе тестирања. Систем на коме је тестирање рађено је ГНУ/Линкус дистрибуција Убунту, верзија 16.04.

3.1 Имплементација Let's Encrypt сертификата на Апач Веб сервер

Let's Encrypt је сертификационо тело које је са радом кренуло у априлу 2016. године, и као што је већ речено, нуди бесплатне X509 сертификате за SSL заштиту. Осим цене, још једна од погодности коју Let's Encrypt нуди је аутоматизован процес издавања, потписивања, инсталације и обнављања (енгл. *renewal*) сертификата, као и избацивање задатка ручне конфигурације Веб сервера из процеса, тако да је јасно зашто је за тек нешто више од годину дана рада ово сертификационо тело достигло бројку од 100 милиона издатих сертификата⁴.



Слика 4. Let's Encrypt CA

Алат који се користи за аутоматизацију у раду са сертификатом је конзолни програм Certbot. Потребан је хостинг где имамо рут приступ - дељени хостинг углавном не долази у обзир. Пошто подразумевани репозиторијуми за АРТ програм на Убунту Линукс дистрибуцији 16.04 не садрже Certbot, потребно је прво додати репозиторијум у АРТ листу извора:

```
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install python-certbot-apache
```

Сада имамо инсталиран Certbot програм на нашем серверу. Следећи корак је прибављање и инсталација сертификата. Како је Апач сервер једно од најпопуларнијих

4 <https://letsencrypt.org/2017/06/28/hundred-million-certs.html>

решења за Веб сервере, већ постоје скрипте које аутоматизују овај посао. Користимо команду:

```
sudo certbot --apache
sudo service apache2 restart
```

Након покретања прве команде у терминалу ће бити приказан псеудо-графички промпт који тражи да унесемо име домена (један или више раздвојених зарезом) и адресу електронске поште власника домена. Такође имамо опцију да укључимо и HSTS заглавље - пожељно је да и ово користимо (опција Secure).

Период важења Let's Encrypt сертификата је јако кратак - 90 дана. Можемо аутоматизовати процес обнове користећи crontab програм. У следећем листингу дат је садржај crontab фајла. **Напомена:** crontab мора бити отворен за рут корисника.

```
...
0 0,12 * * * certbot renew --quiet --post-hook "service apache2 restart"
...
```

Овај процес биће покренут два пута у току дана (у поноћ и у 12 ч.) сваког дана, што је и препорука у документацији.

Симболички (меки) линкови ка кључевима и другим важним фајловима могу се наћи унутар следећег директоријума: `/etc/letsencrypt/live/www.example.com`, где је `www.example.com` име нашег домена. Сви генерисани кључеви су у PEM формату. У случају да је потребна конверзија у неки други формат (бинарни DER или Мајкрософтов PFX формат најчешће) можемо користити OpenSSL програм.

Certbot је написан у Python језику и користи ACME (*Automatic Certificate Management Environment*) протокол за комуникацију са сертификационим телом. Протокол се базира на клијент-сервер архитектури и размени JSON (енгл. *JavaScript Object Notation*) серијализованих порука преко HTTPS протокола. Служи да додатно олакша имплементацију бесплатног SSL сертификата на наш сајт[12]. Постоје разне алтернативе за Certbot, али се, по мишљењу аутора овог рада, издваја Веб сервер Caddy, написан у Go програмском језику који подразумевано нуди безбедну варијанту протокола за пренос хипертекста, без додатних подешавања. Иако представља мали пројекат отвореног кода, може допринети већој револуцији на Интернету - у случају да остали популарни Веб сервери као Апач и nginx обезбеде аутоматизацију у раду са сертификатима, готово да је гарантовано да ће велика већина саобраћаја на Вебу у будућности бити шифрована. У времену када брзина преузимања ресурса са Интернета више не представља проблем у развијеним земљама, јасно је да треба тежити свеprisутности шифроване комуникације.

3.1.1 ACME протокол

ACME или аутоматизовано окружење за управљање сертификатима је протокол апликативног слоја који омогућава да се између сертификационог тела и подносиоца захтева (енгл. *applicant*) аутоматизују процеси верификације и издавања сертификата. Додатно, аутоматизоване су и остале функције, као што је повлачење сертификата. Као што је већ напоменуто протокол комуницира између сервера и клијента преко универзалног JSON формата, тако да је могуће развити сопствено решење за ACME клијента (као замену за Certbot) врло лако у неком програмском језику вишег нивоа. Већ постоји гомила решења отвореног кода на Github репозиторијуму урађених у Python, JavaScript (Node JS), PHP и осталим популарним језицима.

3.2 Правилан избор криптографских алгоритама

Конфигурација SSL протокола на Апач Веб серверу се ради путем његовог додатног модула `mod_ssl`. Рецимо, уколико желимо да ограничимо HTTPS саобраћај тако да подржава само TLS v1.2, можемо користити следећу директиву:

```
SSLEngine On  
SSLProtocol TLSv1.2
```

Популарни CRIME напад о коме ће бити више речи у даљем делу рада је показао да компресија код SSL протокола може направити проблеме, и самим тим, желимо да је искључимо у нашој конфигурацији. Користимо директиву:

```
SSLCompression Off
```

Добар избор криптографских алгоритама такође је од великог значаја за добру конфигурацију SSL протокола на Апачу. RC4 алгоритам је депрактикован званично преко Интернет стандарда⁵, а 3DES је подложен неким познатим нападима (SWEET32). Директива изгледа овако:

```
SSLHonorCipherOrder On  
SSLCipherSuite ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:  
DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS
```

5 <https://tools.ietf.org/html/rfc7465>

Кад разбијемо стринг који је прослеђен другој директиви на под-стрингове између двотачке добијамо:

```
# AES GCM
# + -> move matching ciphers to the current location in list
ECDH+AESGCM
DH+AESGCM
# AES 256
ECDH+AES256
DH+AES256
# AES 128
ECDH+AES128
DH+AES
RSA+AESGCM
RSA+AES
# ! -> kill cipher from list completely
!aNULL # all ciphers using no authentication, <2.4.7 version only
!MD5
!DSS # all ciphers using DSS authentication
```

Овај избор криптографских алгоритама ставља ECDH (Елиптичка крива + Дифи-Хелман алгоритам) заједно са AES симетричном шифром у GCM (енгл. *Galois / Counter Mode*) моду. Овај мод је тренутно актуелан јер је донео побољшане перформансе и ефикасност у односу на дуго фаворизовани CBC (енгл. *Cipher Block Chaining*) мод, као и чувени ECB⁶ (енгл. *Electronic Codebook*) мод.

3.3 Корисна заглавља протокола за пренос хипертекста

Због тога што протокол за пренос хипертекста у безбедној варијанти представља уствари обичан HTTP који се шаље преко SSL протокола, практично цео садржај HTTPS пакета може бити шифрован. Међутим, због потреба TCP протокола, адреса хоста и порт се шаљу у отвореном облику. Узевши у обзир да се преко протокола за пренос хипертекста шаљу креденцијали корисника и слични поверљиви подаци, јасно је зашто је потребно штити комуникацију на Вебу. Нападач може бити у локалној бежичној мрежи са мрежном картицом у мониторинг моду и путем софтвера као што је *Wireshark* једноставно доћи до поверљивих података. Напад може бити озбиљнији, чвор на мрежи између нас и одредишта може да се уметне као човек у средини (енгл. *Man-in-the-middle* - *MITM*). То може бити чак и провајдер Интернет услуге који то ради у маркетиншке сврхе или криминалац који се уметнуо као чвор

6 <https://blog.filippo.io/the-ecb-penguin/>

у оквиру комуникације. Због тога је добра конфигурација Веб сервера битна, како би сузила могућности да дође до нарушавања приватности. Нека од заглавља протокола за пренос хипертекста везана за SSL протокол која се тичу овога су Public-Key-Pins и Strict-Transport-Security.

3.3.1 НРКР механизам

Механизам прикачивања отиска јавног кључа (енгл. *HTTP Public Key Pinning*) се користи за решавање проблема лажног представљања, тј. спречавање нападача да се представи као валидан сервер путем лажног (невалидног) сертификата. То решава тако што доставља клијенту кроз заглавље протокола за пренос хипертекста јавни кључ (или више њих) који могу бити валидни за дати домен.

При укључивању НРКР заглавља потребан је већи ниво разумевања како сам механизам функционише и разумети све опасности. Један од најбољих примера лоше конфигурације код НРКР заглавља је случај када је један познати Веб сајт www.smashingmagazine.com постао недоступан за своје посетиоце на 4 дана (сајт је у првих 5000 по посећености по *Alexa Internet* ранкингу). Пример заглавља које укључује НРКР механизам у одговор сервера је дат у следећем листингу:

```
Public-Key-Pins: max-age=2592000;  
pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";  
pin-sha256="LPJNul+wow4m6DsqxnbnihsWHlwfp0JecwQzYp0LmCQ=";  
report-uri="http://example.com/pkp-report.php";  
includeSubDomains
```

где је:

- *max-age* - број секунди након пријема РКР заглавља за време којих ће клијент (Веб прегледач) чувати сервер запамћен као КРН (енгл. *Known Pinned Host*). Ова директива је обавезна. Вредност ове директиве је цео број (INTEGER).
- *pin-** - обавезна директива која садржи крипто идентитет хоста. Тренутно је једини подржани хеш алгоритам који се користи SHA256 (у овом случају *pin-sha256*). Остављена је могућност да се касније додају нови хеш алгоритми. Вредност је типа стринг и садржи отисак јавног кључа сервера у Base64 формату.
- *report-uri* - Веб страница на коју ће у случају неуспешне валидације НРКР заглавља на страни клијента бити прослеђен извештај о томе. Ово поље је опционо и вредност је стринг.

- *includeSubDomains* - опциона директива која говори претраживачу да се полиса примењује не само за пинованог хоста, већ и за његове под-домене.

Главни проблем код овог заглавља је тај што има превелику одговорност. Као једна од компонената безбедног и комплетно шифрованог Интернета сутрашњице, НРКР на неки начин мења стандардан принцип рада SSL протокола. Без коришћења НРКР заглавља у случају губитка старих кључева можемо једноставно генерисати нови пар кључева и извадити нов сертификат за њих. Међутим, када је кључ пинован, ми у наведеном времену морамо да имамо одговарајући крипто идентитет да би се ресурс са нашег Веб сајта испоручио са сервера на страну клијента нормално. Зато је добро имати пар кључева у резерви (тзв. *backup key*) како би у сваком тренутку конфигурација сервера одговарала и старим посетиоцима нашег сајта, тј. посетиоцима који већ имају пинован крипто идентитет нашег сервера. Такође, један од видова напредних трајних претњи (енгл. *Advanced Persistent Threat - APT*) би био да хакер упадне на наш Веб сервер и укључи НРКР, што је тешко да би администратор приметио одмах. Након неког времена хакер би уклонио пиноване кључеве и захтевао откуп - овакав тип напада се назива RansomPKP (енгл. *ransom* - откуп) и био је представљен по први пут јавности на популарној DEF CON конференцији у Лас Вегасу 2016. године.

Познати амерички хакер Мокси Марлинспик (енгл. *Moxie Marlinspike*), аутор SSL Strip алата, је такође аутор Интернет драфта из 2013. године под називом TACK (енгл. *Trust Assertion for Certificate Keys*). Драфт предлаже пиновање посебних, TSK (енгл. *TACK signing key*) кључева, уместо кључева сертификационог тела. Иако пројекат олакшава имплементацију механизма пиновања и додатно удаљава сам процес имплементације од сертификационог тела, након што драфт није прошао и сам пројекат је престао бити активан.

3.3.2 HSTS механизам

Полиса о строгој безбедности транспорта (енгл. *HTTP Strict Transport Security Policy*) решава проблем напада као што су уназађивање протокола (енгл. *Protocol downgrade attack*) и крађу колачића (енгл. *cookie-hijacking*). То се постиже тако што се забрањује дељење ресурса обичним протоколом за пренос хипертекста, већ се дозвољава искључиво коришћење безбедне верзије протокола. Заглавље које укључује HSTS механизам у одговор сервера би могло да изгледа овако:

```
Strict-Transport-Security: max-age=16070400; includeSubDomains
```

Примећујемо да су директиве исте као и код НРКР механизма. Иако не решавају исти проблем, јасно је да је HSTS много елегантније решење за проблематику коју решава. То може бити и проблем код НРКР заглавља, као и разлог зашто је у пракси ређе коришћен од HSTS заглавља.

Овакве видове заштите је могуће реализовати и на нивоу саме апликације⁷, али је реализовањем на нивоу протокола сам процес имплементације знатно олакшан. Уколико користимо Certbot програм на начин на који је наведен у претходном делу рада, HSTS заглавље ће бити аутоматски укључено. Истовремено, могуће је и укључити га ручно користећи нпр. Апачов `mod_headers`.

3.3.3 Остала корисна заглавља и OWASP препоруке

OWASP⁸ је онлајн заједница која се бави објављивањем бесплатних чланака, методологија, документација, алата и технологија у области заштите Веб апликација. Основана је 2001. године. Неки од значајнијих пројеката OWASP организације су:

- OWASP Top 10 - идентификује најризичније нападе на Веб апликације и објављује их редовно у виду ранг листе
- Пројекти у области безбедног развоја Веб апликација:
 - OWASP SAMM (енгл. Software Assurance Maturity Model)
 - OWASP Development Guide
- Пројекти у области пентестинга:
 - OWASP Testing Guide
 - OWASP Code Review Guide
 - OWASP ZAP (енгл. Zed Attack Proxy) - алат за откривање пропуста у Веб апликацијама, алтернатива за много популарнији Burp Suite алат
 - Webgoat - рањива Веб апликација намењена за потребе учења пентестинга

OWASP такође има секцију где нуди савете за правилну имплементацију SSL протокола на Вебу[13]. Савети су следећи:

- Коришћење SSL заштите свуда - није довољно само ограничити приступ интерне мреже
- Забрана комуникације са сервером у отвореном (нешифрованом) облику
- Коришћење безбедних колачића (*Secure* опција)
- Држање поверљивих података ван униформног локатора ресурса - иако SSL обелодањује само име домаћина а не цео URL, URL може бити обелодањен кроз историју претраживања унутар прегледача или *Referral* заглавље

⁷ <https://github.com/riamar/pubkey-pin-android>

⁸ <https://www.owasp.org/>

- Забрана кеширања (*Cache-Control: no-cache, no-store* и *Expires: 0*)
- Коришћење HSTS и HPKP механизма
- Коришћење јаких кључева и заштита истих
- Коришћење сертификата са одговарајућим доменом - ово је могуће на два начина:
 - Коришћење једног сертификата за abc.example.com и другог за xyz.example.com
 - Коришћење заједничког сертификата за оба под-домена, где би заједнички CN био example.com, док би поље Subject Alternate Names - SAN садржало под-домене abc.example.com и xyz.example.com
- Без коришћења џокер (*) симбола у сертификатима
- Миграција са SHA-1 на SHA-256 - ово је још битније сада када су напади на SHA-1 објављени⁹
- Коришћење само јаких верзија SSL протокола (од TLS v1)
- Имплементација PFS - сесијски (ефемерни) кључеви
- Омогућавање коришћења искључиво јаких крипто алгоритама
- Искључивање компресије
- Подршка за TLS-PSK
- Коришћење ажуране верзије крипто библиотеке (openssl)

3.4 Механизми за информисање о статусу ревокације код X509 сертификата

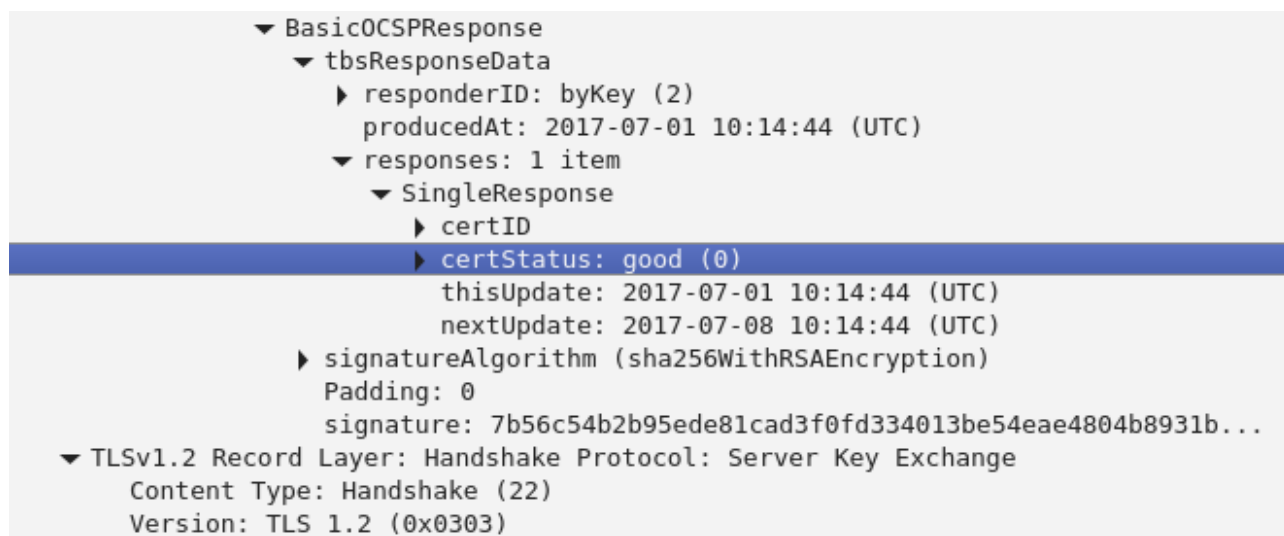
Кад Веб читач успостави конекцију са Веб сајтом који користи SSL протокол за заштиту, дигитални сертификат Веб сервера мора да прође кроз пар провера. Једна од њих је провера да ли се Веб сертификат налази на листи за повлачење сертификата (енгл. *Certificate Revocation List - CRL*).

Многи су разлози за повлачење (ревокацију) сертификата - то може бити због неког безбедносног пропуста услед кога долази до компромитовања приватног кључа или чак због лоше генерисаног сертификата од стране сертификационог тела. CRL представљају листе које садрже серијски број повучених сертификата и датум повлачења. CRL функционишу по принципу црне листе (енгл. *blacklist*) и чувају се кеширане на страни клијента - ово нападачу оствља (мањи) временски период за који може да протури повучени сертификат као активан и даље, јер се ажурирање CRL не обавља у реалном времену (енгл. *real-time*).

⁹ <https://shattered.io/>

Да би се спречила могућност од оваквих напада, донесен је нови Интернет стандард - OSCP (енгл. *Online Certificate Status Protocol*). Овај протокол функционише тако што се листа не кешира, већ се при добијању сертификата Веб сервера информација о његовој могућој ревокацији испитује директно код сертификационог тела. Међутим, овде постоји проблем што сертификационо тело онда може имати увид у то које сајтове клијенти посећују. Могуће решење за то донео је Савет за безбедност сертификационих тела (енгл. *Certificate Authority Security Council*), чији су чланови нека од водећих сертификационих тела.

OCSP спајање (енгл. *OCSP stapling*) представља заглавље које се додаје одговору Веб сервера и садржи OSCP одговор сертификационог тела. Ово штити приватност клијента и повећава брзину испоруке садржаја сајта због тога што се шаље у виду заглавља протокола, а не као додатан упит клијента.



Слика 5. OCSP одговор може бити "good", "revoked" или "unknown"

Код Апач Веб сервера OCSP спајање је подржано од верзије 2.3.3[14]. Прво је потребно набавити сертификат сертификационог тела које нам је издало наш сертификат у PEM формату. Сертификат за Let's Encrypt CA је доступан на страници <https://letsencrypt.org/certificates>. Када набавимо сертификат можемо га преузети користећи wget алатку:

```
wget https://letsencrypt.org/certs/isrgrootx1.pem.txt -O ca-certs.pem
```

Затим отварамо конфигурациони фајл нашег сајта путем неког псеудо-графичког едитора (GNU nano, Vim, Joe и сл.) и укључујемо следеће директиве:

```
SSLCACertificateFile /etc/ssl/ca-certs.pem
SSLUseStapling on
```

Након овог наш сајт осим што обезбеђује посетиоцима безбедност путем OCSP стандарда, обезбеђује им и додатан вид приватности јер не морају слати упит сертификационом телу за статус нашег сертификата нашег.

3.5 Заштита приватних кључева

Код SSL протокола заштита почиње са крипто идентитетом сервера. Потребан је јак приватни кључ који би спречио нападаче да се представљају у наше име, али је такође потребан и валидан сертификат који би повезао тај кључ са жељеним доменом. Ове две ствари представљају фундаменталне основе свега осталог код SSL протокола - без ове две ствари не значи нам ни правилно конфигурисани HPKP, HSTS и OCPS механизми.

Не постоји универзалан одговор на питање колика треба бити величина кључа[15]. При избору величине кључа, два су главна параметра на која треба обратити пажњу. Први је вредност тога што чувамо - листа клијената неке омање компаније може вредети 1000 долара, док маркетинг подаци неке велике компаније могу вредети 1000 пута више. Други параметар је време које је потребно штитити ту вредност. Нека медијска кућа има потребу да штити податке само један дан - данашња тајна ће свакако бити наслов сутра. С друге стране, информације о развоју неког производа морају остати тајне све док развој траје - можда и годинама.

Препоруке за минималну величину кључа су 2048 бита, док NIST упозорава да ни то није довољно. Нека сертификациона тела нуде да вам и генеришу кључеве при издавању сертификата, што логично у старту нарушава приватност кључа. Обнављање (енгл. *renewal*) кључа треба вршити макар једном годишње, ако не и чешће у случају да је процес аутоматизован (што Certbot омогућава). У случају да нас HPKP не спречава у томе, препоручено је да генеришемо нови пар кључева при сваком захтеву за потписом сертификата. При најмањем компромитовању кључа препоручује се повлачење сертификата и генерисање нових кључева.

Иако OpenSSL при генерисању кључева нуди могућност укључивања лозинке која би служила за заштиту приватног кључа, ово није поуздано решење. Код организација које захтевају најстрожије безбедносне критеријуме, користе се тзв. хардверски безбедносни модули (енгл. *Hardware security modules - HSM*) - физички рачунарски уређаји који служе за заштиту дигиталних кључева. Традиционално се производе у форми екстерног уређаја који се прикључује директно на рачунар. Један од најпопуларнијих производа из ове категорије на тржишту представља nShield серија француске компаније Thales¹⁰. Уређај поседује јединицу за напајање, USB порт, Ethernet порт, мини дисплеј, LED индикатор и интерфејс за читање смарт картица. При генерисању кључа који ће бити складиштен на овом уређају потребно је такође спровести додатне мере опреза - препоручује се да се генерисање обави на машини која нема приступ Интернету како би опсег претњи био значајно смањен.

10 <https://www.thalesecurity.com/products/general-purpose-hsms/nshield-connect>



Слика 6: Thales nShield безбедносни уређај

Осим HSM уређаја, као алтернатива за складиштење и преношење кључева се појављују и смарт картице, чија је главна предност могућност интерне обраде података - овиме се може омогућити да кључеви никада не напусте картице, или да не напусте картице ван периода у којем је њихова употреба одобрена[16].

3.6 Остали методи заштите

HTTPS Everywhere¹¹ је још један пројекат EFF организације (енгл. *Electronic Frontier Foundation*, уз Мозила фондацију оснивачи Let's Encrypt пројекта, прим. аут.). Овај пут ради се о проширењу за популарне Веб прегледаче - Google Chrome и Mozilla Firefox. Такође се налази преинсталиран у Тор прегледачу. Проширење ради тако што за сваки линк који треба да учита Веб прегледач прво провери да ли постоји HTTPS верзија ако већ није. Иако је концепт одличан, овакав начин може довести до већих успоравања при учитавању Веб сајтова. Представља корисно решење за заштиту на страни клијента.

Опортунистички TLS (енгл. *Opportunistic TLS*) представља проширење у обичним, текстуалним комуникационим протоколима, где се у случају да постоји шифрована верзија, прелази на њу (отуда и назив опортунистички). Представља добру алтернативу за HTST на сервису електронске поште. Функционише тако што клијент и сервер нормално започињу комуникацију преко отвореног протокола, затим клијент издаје наредбу STARTTLS. У случају да сервер поседује могућност шифроване комуникације, одговара са наредбом 250 STARTTLS. Након тога прелазе на шифровану (SSL) комуникацију. Синтакса не мора бити иста за све протоколе, али протоколи који подржавају опортунистички TLS су између осталог IMAP, POP3, SMTP, XMPP, NNTP, IRC, LDAP као и FTP преко наредбе AUTH TLS.

PGP (енгл. *Pretty Good Privacy*) представља добру алтернативу за заштиту приватности и аутентификацију при размени података. Међутим, фундаментална разлика између PGP програма и SSL протокола је та што PGP ради на ускладиштеним подацима, док SSL ради са подацима који су у фази транспорта. PGP тешко може да замени SSL на Вебу, али нпр. код

11 <https://www.eff.org/https-everywhere>

FTP (енгл. *File Transfer Protocol*) протокола, који има две безбедне варијанте - SFTP (FTP преко SSH) и FTPS (FTP преко SSL-a), и обе варијанте су значајно спорије од обичне, PGP може имати предности у односу на SSL. У овом случају, подаци се шифрују локално а затим шифровани шаљу обичном варијантом FTP протокола. Протокол који се бави заштитом приватности и аутентификацијом код сервиса електронске поште је OpenPGP. Представља Интернет стандард[17] и разликује се од PGP-a по питању лиценце - PGP је власнички софтвер. Алтернатива за PGP за коришћење OpenPGP протокола је GPG (енгл. *GNU Privacy Guard*).

Тор је већ поменут у овом раду. Представља сервис који обезбеђује анонимност на Интернету тако што рутира Интернет саобраћај клијента кроз мрежу волонтера уместо директно на одредиште[18]. То значи да наш рачунар када желимо да приступимо страници на адреси www.google.rs уместо да шаље захтев ка страници кроз провајдер он од провајдера захтева да посети страницу првог чвора у мрежи Тор волонтера, која функционише као *peer-to-peer* мрежа. Тор мрежа користи шифровање на апликативног нивоу за заштиту података - што значи да може да се користи без SSL протокола. Међутим, Фејсбук је 2014. године урадио занимљиву ствар - понудио је свој сервис на Торовој мрежи, и уз то посебан сертификат потписан од стране Digicert сертификационог тела за његов .onion домен. Ово отвара нова питања у SSL свету, јер .onion домени нису стандардни домени највишег нивоа (енгл. *top-level domain* - *TLD*). Тренутно, .onion сервиси нису толико заступљени у пракси (барем не у легалне сврхе) да би се журило са решењем.

4 Напади на SSL протокол

У фебруару 2015. године, међународна организација под називом Специјална комисија за развој Интернета (енгл. *Internet Engineering Task Force - IETF*), објавила је RFC документ[19] који описује већ познате нападе на SSL протокол. Напади су категоризовани у неколико категорија:

- Напади поновног преговарања (енгл. *renegotiation attacks*)
- Напади уназађивањем протокола (енгл. *downgrade attacks*) - FREAK и Logjam напади
- Напади преко унакрсних протокола (енгл. *cross-protocol attacks*) - DROWN напади
- Експлоатација преко Веб прегледача - BEAST
- Напади базирани на проблемима са компресијом - CRIME и BREACH напади
- *Padding oracle* и POODLE напади
- Пропусти у имплементацији: *Heartbleed*, *BERserk* и *Cloudflare* пропусти
- Разни напади на RC4 протокол

4.1 Heartbleed напад

У претходном делу рада је поменуто да је за имплементацију SSL протокола убедљиво најчешће коришћена OpenSSL библиотека. Када један софтвер држи толики примат у својој категорији, сигурносни пропусти у њему су критичнији због тога што више машина може бити изложено нападу. У априлу 2014. године пронађен је пропуст у OpenSSL библиотеци који дозвољава нападачу да ишчита садржај меморије сервера који је заштићен рањивом верзијом библиотеке. Исти дан када је пропуст објављен отклоњен је из библиотеке. Проблем је што је верзија OpenSSL библиотеке са тим пропустом објављена још 2012. године. То значи да је пропуст био неотклоњен читаве 2 године - тзв. *0-day exploit*. Већ поменути *Bloomberg News* је оптужио америчку државну агенцију за безбедност (NSA) да је знала за пропуст још 2012. када се појавио у софтверу. Пропуст је направио немачки програмер, али треба имати у виду да је OpenSSL слободан софтвер отвореног кода који одржава заједница, тако да је право чудо да је пропуст отклоњен тек после читаве две године. Неки научници чак и тврде да нападачи нису искористили оволико критичан пропуст онолико колико су била почетна предвиђања[20]. С друге стране, овакви случајеви могу и неким компанијама дати повод да пређу са софтвера отвореног кода на неко комерцијално решење затвореног кода верујући да ће бити безбедније, што нема претерано везе са реалношћу што најбоље доказује криптологија, где су шифарски алгоритми доступни нападачу.

Heartbleed пропуст (CVE-2014-0160) омогућава нападачу да извуче потенцијално поверљиве податке директно из меморије (до 64 KB података по одговору). Након малициозног захтева, на серверу не остаје никакав траг од нападача. Могуће је експлоатисати пропуст и на серверу и на клијенту - преко 2/3 Веб сервера користи OpenSSL имплементацију, док је на клијентској страни најкритичнији пример оперативни систем за мобилне телефоне компаније Гугл Андроид (заражена верзија 4.1.1). Heartbleed пропуст је пропуст специфичан за OpenSSL библиотеку, нема везе са пропустом у архитектури SSL протокола.

CVE-2014-0160 је званична референца на Heartbleed пропуст. CVE (енгл. Common Vulnerabilities and Exposures)¹² је стандард референцирања пропуста и рањивости у софтверу. Сваки познати пропуст има свој идентификатор - CVE-ID. Организација која спроводи CVE стандардизацију је MITRE организација.

Heartbleed користи тзв. *heartbeat* механизам унутар OpenSSL имплементација протокола (отуда и назив *Heart-bleed*).

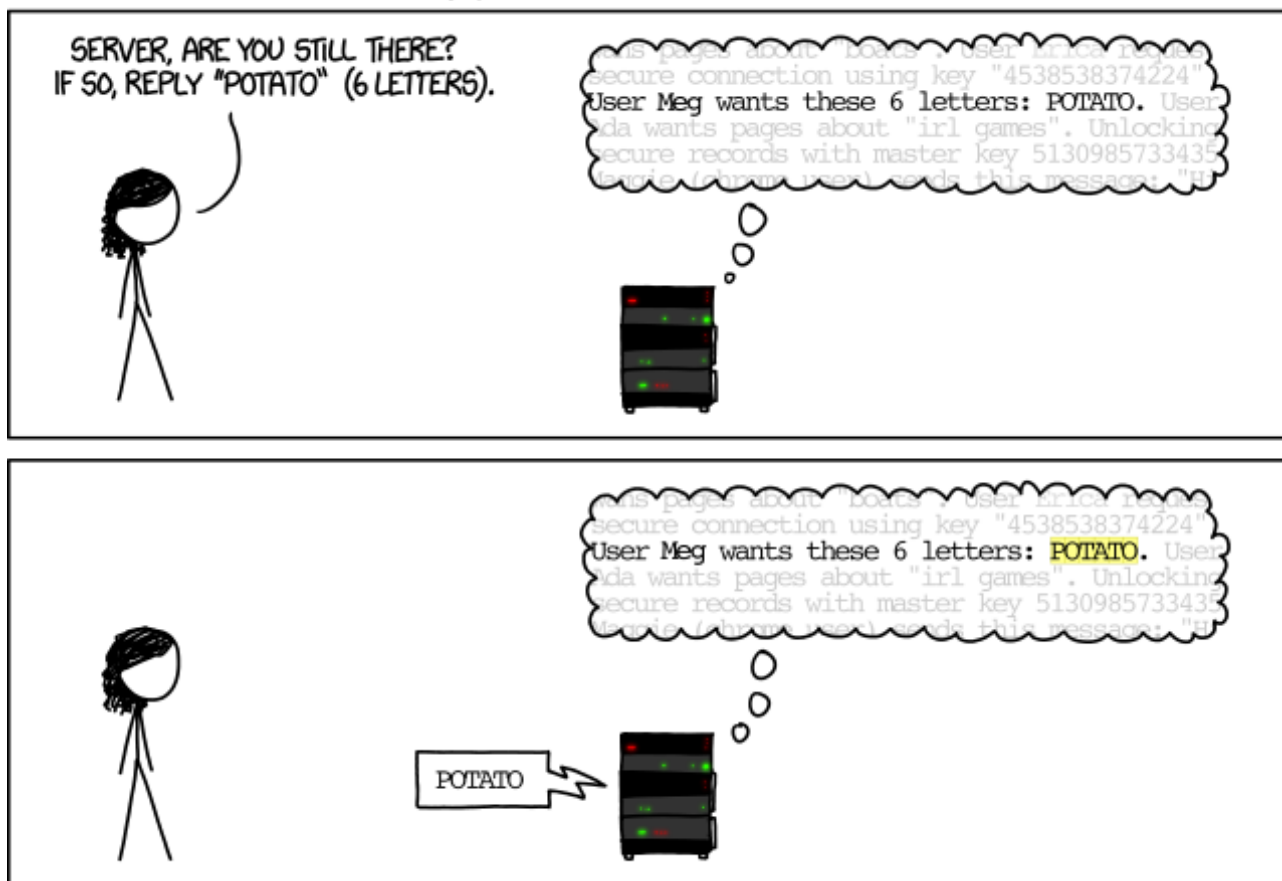
Рањивост Веб-сајта на Heartbleed пропуст можемо проверити на више начина. Популарни оквир за тестирање пробојности Метасплоит¹³, доступан и у комерцијалној и у бесплатној верзији, као и мрежни скенер Nmap кроз NSE (енгл. *Nmap Scripting Engine - NSE*) садрже скриптове који могу да детектују (и експлоатишу) Heartbleed пропуст. Рањивост на Heartbleed можемо проверити такође и путем Веба, један од примера за то је бесплатан сервис компаније High-Tech Bridge, доступан на локацији: <https://www.htbridge.com/ssl>.

4.2 Како Heartbleed ради

Heartbleed напад је стриктно везан за коришћење SSL протокола, али не само за Веб сервис - SSL је свеprisутан на Интернету као најпопуларнији протокол нижег слоја са шифровање саобраћаја. Користи се и за сервис електронске поште, шифровање протокола за пренос фајлова (енгл. *File Transfer Protocol - FTP*), безбедну комуникацију између сокета итд. Познато је да се Веб сервис, тачније HTTP протокол, заснивају на клијент-сервер архитектури. Принцип код сваког протокола који користи клијент-сервер архитектуру је тај да сервер опслужује клијента, најчешће на клијентов захтев - у случају Веб сервиса клијент захтева удаљени ресурс (нпр. фајл *index.html*) и сервер му одговара са статусом 200, садржајем фајла и пратећим заглављима.

¹² <http://cve.mitre.org/>

¹³ <https://www.metasploit.com/>



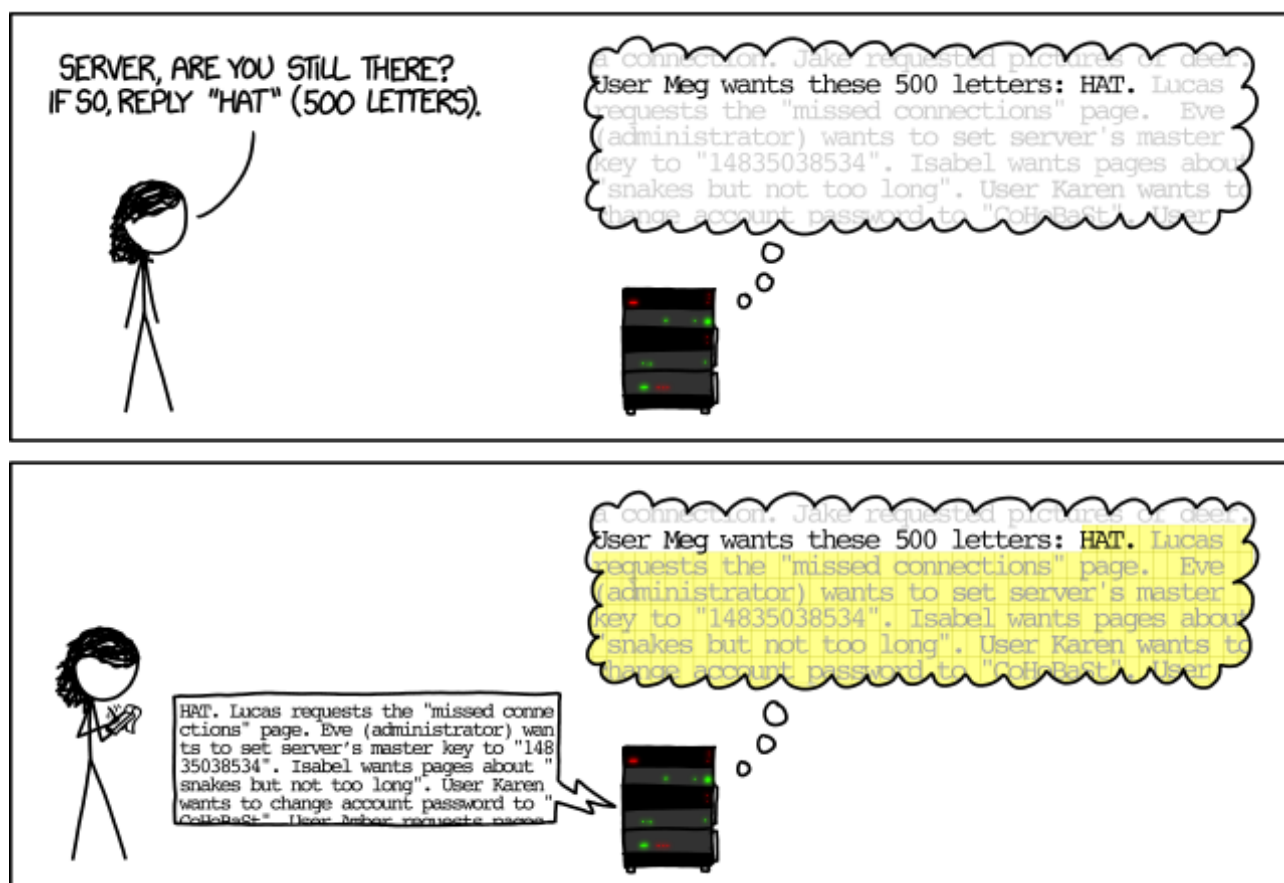
Слика 7. У овом случају клијент захтева од сервера да одговори са стрингом "POTATO" дужине 6 карактера ако је доступан. Рендал Монро, *xkcd.com*.

Heartbeat функционалност која се користи у оствареној SSL комуникацији се понаша баш као небезбедни клијент-сервер протокол на слици 2. Намена је једноставна, heartbeat треба да чува конекцију отвореном чак и у случају да се подаци не преносе тренутно - ово је корисно због скувих операција при успостављању везе (генерисање сесијског кључа, аутентификација), наспрам криптографски једноставнијих операција (симетрично шифровање). "Нормалан" начин рада подразумева да клијент пошаље пакет серверу који садржи секвенцу бајтова и дужину секвенце. Сервер одговара истом том секвенцом.

OpenSSL је open-source пројекат, што значи да свако може да учествује у развоју. Користи Гит систем за верзионисање и хостују сопствени јавни репозиторијум¹⁴. Програмери који учествују у развоју прво пошаљу кођ који треба да прође неколико провера да би касније био комитован. Heartbeat функционалност је имплементирана од стране немачког програмера Робина Сигелмана, док је последњу проверу и комит одрадио др Стефан Хенсон. Занимљиви су време и датум комита: 3603 секунде пре Нове године 2012.

¹⁴ <https://git.openssl.org/gitweb>

Проблем је настао у томе што је фалила провера на серверској страни, где упоређује дужину секвенце бајтова коју је клијент послао и стварну дужину секвенце. Нпр. ако клијент генерише секвенцу од 1 бајта а затим параметру који означава дужину секвенце проследи дужину 64 KB, сервер ће вратити 1 бајт из секвенце и ишчитати још 64 KB - 1 B из своје радне (RAM) меморије. Пошто се у радној меморији чувају подаци са којима рачунар тренутно ради, нападач може очекивати да добије креденцијале корисника, SQL упите, чак и приватне кључеве на серверу. У случају да сервер има више сервиса, од којих је само један рањив на Heartbleed, и остали сервиси би били у опасности јер деле исту радну меморију.



Слика 8. У другом кораку клијент захтева од сервера да одговори са стрингом "HAT" дужине 500 карактера. Уместо да одбаца захтев, сервер одговара са садржајем из радне меморије. Рендал Монро, xkcd.com.

4.2.1 Ишчитавање садржаја радне меморије

Постоји више начина на које нападач може ишчитати садржај радне меморије сервера и евентуално доћи до поверљивих података. Осим поменутог Heartbleed пропуста, неки од вектора напада су и:

- *Cold boot attack*, где нападач има физички приступ машини

- *Direct Memory Access* напад, где нападач експлоатише портове који дозвољавају директан приступ меморији (нпр. *FireWire* и *Thunderbolt*)
- Програми који користе */proc/self/mem* за читање меморије - не толико стандардно и још мање лако
- Када се садржај радне меморије нађе на спољној меморији - чест случај:
 - Нпр. када је капацитет радне меморије скоро испуњен, подаци се чувају на спољној меморији (своповање, енгл. *swapping*)
 - Приликом хибернације, садржај радне меморије се сачува на спољној меморији да би касније био учитан назад

Јасно је да је приступ меморије нападачу озбиљан проблем - програм који ради са лозинкама мора да их чува у меморији макар на кратак период времена. Међутим, постоји могућност да одмах кад програм заврши рад са лозинком, уклонимо лозинку из меморије. Ово смањује временски простор нападачу у којем може да ишчита лозинку из меморије. Свакако, ако нападач има приступ радној меморији, не треба да се бринемо само око лозинки, али размотримо како можемо повећати безбедност брисањем поверљивих бајтова из меморије одмах након завршетка рада са њима.

Код многих програмских језика није толико лако очистити садржај меморије. Имамо пример језика ниског нивоа као што је С који директно ради са меморијом. Када дефинишемо стринг у С језику, директно радимо са меморијом - `char [ ]`. Једном када очистимо садржај дефинисаног низа карактера (сетујемо вредност на празан карактер нпр.), лозинка ће нестати из меморије. Сличан случај је и код Јава програмског језика, који није толико ниског нивоа. Али, Веб апликације користе интерпретиране језике високог нивоа као што су PHP и Python. Ту овај процес није толико лак, и обично је беспотребан. Размотримо пример Python језика.

Постоји неколико начина да наш програм прочита меморију:

- Читање */proc/self/mem* фајла или */proc/*PID*/mem* - овако најчешће раде дебагери
- Генерисање тзв. *core* фајла користећи *gcore* алат

Први начин је компликован, потребно је преко фајла */proc/\$pid/maps* извући меморијски опсег процеса и затим приступити наведеном фајлу само у том опсегу, све као рут корисник. Пошто је *gcore* програм лако доступан на већини Линукс машина, користећемо други приступ. Размотримо следећих пар линија Python кода:

```
with open('key.txt', 'r') as fr:
    secret = fr.read()
# end

print(secret)
```

```
import os; print('ПИД:', os.getpid())

input() # Паузирамо програм
```

Фајл *key.txt* је обичан текстуални фајл који садржи нашу шифру у једној линији ('tajna'). Програм учитава текстуални фајл са нашом шифром, чита шифру у променљиву *secret* и затим штампа шифру на стандардном излазу. Последње две линије су додате за потребе демонстрације, прва нам исписује идентификатор покренутог процеса (алтернатива *'pgrep python3'*), друга паузира извршавање програма. У мом случају програму је додељен PID 32542. Након тога покрећемо *gcore*:

```
sudo gcore 32542
```

Ако је све прошло како треба приметићемо да је генерисан фајл *core.32542*. Погледаћемо да ли је шифра 'tajna' садржана у том фајлу.

```
grep 'tajna' core.32542
Binary file core.32542 matches
```

Додајмо једну линију у скрипту изнад позива *print* функције:

```
print(secret)
secret = None
```

Сада је програму додељен PID 32688:

```
sudo gcore 32688
grep 'tajna' core.32688
```

Сада стринг 'tajna' није више остао у меморији! Треба напоменути да старије верзије Python језика (цела 2 и 3 донекле) нису имале овакво понашање, па због тога старији модули који се користе у сврхе безбедности као што је *PyCrypto* модул користе *byte* или *bytearray* структуре уместо стрингова.

Као што је напоменуто изнад, за "нормалне" Веб апликације генерално није потребно ићи оволико у крајност када се ради на повећању безбедности. Довољно је имати валидан SSL сертификат и пратити OWASP препоруке за безбедно кодирање¹⁵.

15 https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide



Слика 9. CVE-2014-0160

4.3 Padding oracle напад

Крипто алгоритми раде са блоковима фиксне дужине, међутим, не може дужина сваке поруке бити дељива са величином блока, што значи да ће у последњем блоку најчешће остати мање бајтова од онога што се користи у блоку. Због тога допуњавамо последњи блок до задате дужине секвенцом која се назива падинг (енгл. *padding*). *Padding oracle* напад се базира на томе да нападач поседује један шифрат, али и на томе да се сервер понаша као оракл, тј. да на сваки шифрат који стигне шаље одговор да ли је пристигла порука валидно допуњена или не. Овај напад се углавном везује за симетричне блоковске алгоритме, али је могућ и у асиметричној криптографији[21].

4.3.1 Објашњење напада, CBC мод и криптографски оракл

CBC (енгл. *Cipher Block Chaining*) представља један од начина на који се шифрују блокови код симетричних шифарских алгоритама. Безбеднији је од првобитног ECB (енгл. *Electronic Codebook*) мода код кога се сваки блок шифрује појединачно - чувени ECB пингвин¹⁶. Ипак, данас је безбедносна препорука CFB (енгл. *Cipher Feedback*), варијација CBC мода. Основне операције код CBC мода:

- Шифровање: $C_i = E_k(P_i \text{ XOR } C_{i-1}), C_0 = IV$
- Дешифровање: $P_i = D_k(C_i) \text{ XOR } C_{i-1}, C_0 = IV$

Допуњавање се ради сваки пут када је последњи блок отвореног текста краћи од дужине блока (дужина блока једнака је дужини иницијализационог вектора). Нпр. ако је дужина блока 16, а последњи блок отвореног текста 12 стринг од 12 карактера, допуњавамо

¹⁶ <https://blog.filippo.io/the-ecb-penguin/>

са 3 произвольна карактера и последњим четвртим карактером који представља дужину падинга (у овом случају $\text{dec}(4) = \text{x04}$):

[K|R|A|J|P|O|R|U|K|E|.|\xFF|\xFF|\xFF|\xFF]

У случају да је последњи блок дељив са 16, допуњавамо са 15 произвольних $\backslash\text{FF}$ бајтова, и последњим који представља индикатор - x10 , $\text{dec}(16)$. Након што се порука дешифрује, прималац тримује вредност добијену у последњем блоку са десне стране и порука је очишћена од допунских карактера.

Криптографски оракл представља механизам на страни примаоца који при примању поруке гледа да ли је падинг валидан или не, и одговара нам, нпр. порука 0 ако је падинг валидан, односно 1 ако није.

Тежи део напада је сам предуслов, нападач мора да буде у поседству бар једног шифрата из актуелне сесије. Претпоставићемо да нападач већ поседује шифрат, и да се на серверу налази криптографски оракл. Шифрат је такође подељен по блоковима: $C_0, C_1 \dots C_i$. Нападач може модификовати, нпр. последњи бајт другог блока (C_1) и пратити шта се дешава кад пошаље то ораклу. Модификација последњег бајта другог блока би изгледала овако:

$$C'_1 = C_0 \text{ XOR } 000000001 \text{ XOR } g$$

где g представља случајан бајт из опсега $[\text{x00}, \text{xFF}]$. Уколико је g исправан, тј. одговара последњем бајту другог блока, g и стварни последњи бајт ће поништити један другог приликом дешифровања и последњи бајт ће бити x01 . Како падинг дужине $\text{hex}(\text{x01}) = 1$ треба да буде секвенца "x01" , добићемо 0 (валидан падинг) као одговор! Исти процес можемо урадити и за претпоследњи бајт, само би уместо x01 користили $\text{xFF}\text{x02}$, за трећи бајт од позади користили би $\text{xFF}\text{xFF}\text{x03}$ итд. Како је блок састављен од 16 бајтова, то значи да би нам у најгорем случају за дешифровање читавог блока C_1 било потребно највише $16 * 255$ (број могућих карактера, $[\text{x00}-\text{xFF}]$), што је прилично тривијално за Интернет.

Треба напоменути још једну битну претпоставку која мора бити испуњена да би напад био могућ - MAC (енгл. *Message Authentication Code*) провера мора бити одрађена пре шифровања на страни пошиљаоца. Уколико би страна примаоца (на којој се налази оракл) добила шифрат и MAC кôд шифрата, могла би да провери MAC пре дешифровања и одбаци поруку као невалидну у случају да је стигла од треће стране (нападача). Дакле, препорука за симетрично шифровање је: шифровање-па-MAC, уместо MAC-па-шифровање. Такође, напад зависи и од одговора оракла који је довољно информативан за нападача - код OpenSSL имплементације конкретно ово нису биле поруке 0 (валидан падинг) или 1 (погрешан падинг), већ временски интервал у којем оракл одговара - у случају бржег одговора нападач би знао да је падинг погрешан. Такође, у случају коришћења CTR (енгл. *Counter*) мода уместо CBC или CFB модова, овај напад не би био могућ, јер падинг ни не постоји.

Напад *padding oracle* осмислио је француски криптоаналитичар Серж Водене (енгл. *Serge Vaudenay*), један од главних у швајцарској LASEC (енгл. *Laboratory of Security and Cryptography*) организацији, која је се прославила по бесплатном *open-source* програму

Ophcrack који се користи за разбијање (крековање) корисничких лозинки на MS Windows оперативним системима.

Старије верзије познатог клијента за DRM видео игре, *Steam*, развијеног од стране *Valve* корпорације, рањиве су на овај тип напада.¹⁷

4.3.2 Варијације на Padding oracle напад

Од 2015. године SSL 3.0 је депрактикован у корист TLS протокола због POODLE (енгл. *Padding Oracle On Downgraded Legacy Encryption*) напада. Напад је презентован у научном раду Гугловог безбедносног тима у септембру 2014. године, да би рањивост била отклоњена месец дана касније. Крајем исте године појавила се и POODLE TLS варијанта, која захтева мање корака у односу на оригиналан напад, а експлоатише неправилну валидацију падинга код неких TLS имплементација.

Lucky Thirteen напад је откривен на Лондонском Универзитету 2013. године и представља временски напад (енгл. *timing attack*). Код временског напада, нападач анализира време потребно за извршавање крипто алгоритама да би компромитовао крипто систем[22]. OpenSSL, као и GnuTLS, су биле рањиве на овај напад. Рањивост са CVE-ID 2016-2107 идентификатором је открила да се приликом крпљења пропуста који је омогућавао *Lucky Thirteen* напад јавио још један Padding oracle напад код OpenSSL библиотеке.

Рањивост на CVE-ID 2016-2107 и POODLE преко TLS протокола могу бити детектовани на бесплатном сервису компаније High-Tech Bridge на локацији: <https://www.htbridge.com/ssl>.

4.4 Напади преко бочних канала - проблеми са компресијом

У криптографији, термин "напад преко бочних канала" представља напад који се базира на информације прибављене преко физичке имплементације самог криптосистема, а не користећи метод сирове силе (енгл. *brute force*) или теоријске пропусте у самом крипто алгоритму (криптоанализа) - нпр. брзина одговора сервера може пружити неке информације нападачу о комуникацији[23].

Велику групу напада на SSL протокол чине напади који се фокусирају на бочне нападе преко компресије у SSL протоколу.

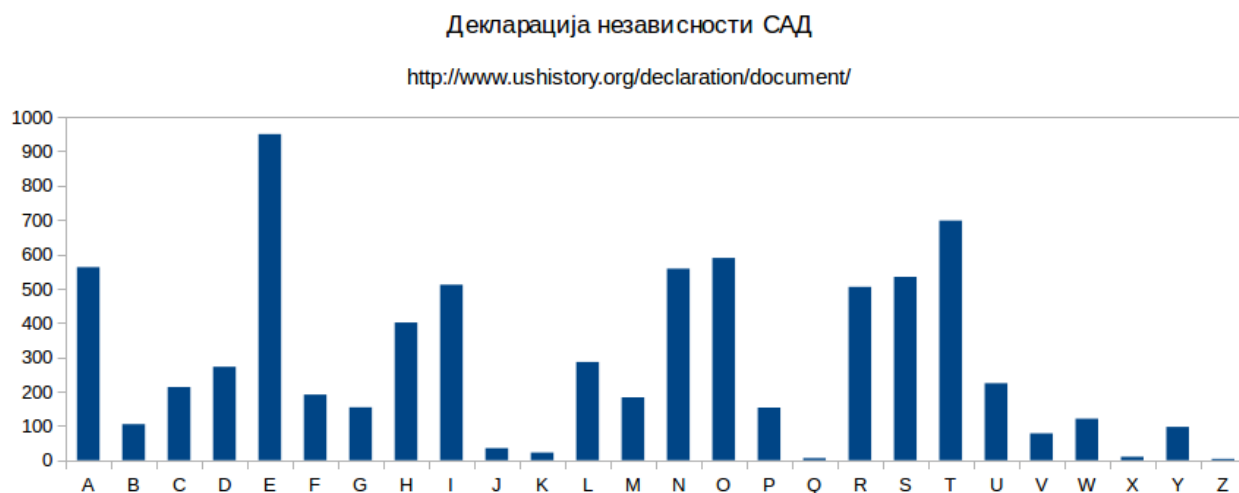
¹⁷ <https://steamdb.info/blog/breaking-steam-client-cryptography/>

4.4.1 Алгоритам компресије

Алгоритме компресије можемо поделити у две групе:

- Компресија преко ентропије
- Компресија преко речника

Идеја код компресије преко ентропије је да се симболи који се понављају чешће од осталих представљају са мање бита од осталих - пример овакве компресије је Хафманово кодовање (енгл. *Huffman coding*). Оваква идеја није потекла из информатике, ако погледамо Морзеову азбуку (енгл. *Morse code*) насталу у XIX веку за потребе преношења сигнала преко електричних телеграфа, видећемо да су најчешће коришћена слова из енглеског алфавета, Е и Т, представљени као тачка [.] и цртица [-] респективно.



Слика 10. Фреквенције слова у "Декларацији независности" (Сједињене Америчке Државе, 1776. година)

Компресија преко речника се базира на идеји да се одређене фразе (може бити и само једна реч) помињу више од осталих у документу. Нпр. ако би претражили фразу "SSL протокол" у овом раду, приметили би да се помиње више од осталих. Компресија преко речника решава ово тако што уместо да испише "SSL протокол" следећи пут само остави референцу на прво појављивање фразе. Пример алгоритма који користи ову компресију је LZ77, који користи свеприсутни GNU алат *gzip*. Размотримо још један пример пре него што демонстрирамо како ово нападач може експлоатисати.

Рецимо да имамо фразу "Ако човек сваког дана стоји на ивици и гледа у амбис, једног дана ће и амбис погледати у њега." и да желимо да је компресујемо помоћу речника. У тренутку пре компресије, тај стринг има 94 карактера. Примећујемо да се одређене фразе понављају ("дана", "амбис", "гледа"). Након компресије стринг ће изгледати овако: "Ако човек сваког дана¹ стоји на ивици и гледа² у амбис³, једног ¹ ће и ³ по²ти у њега.", што је за 11 карактера краће од оригинала. Поента је да што се више фраза понавља, величина компресованог стринга ће бити мања. Како ово нападач може искористити?

Ако текст поруке (нпр. HTTPS одговор) садржи и тајни параметар и параметар који дефинише корисник, можемо погодити тајни параметар само пратећи величину компресованог резултата. Рецимо да Веб апликација којој приступамо користи шифроване и компресоване HTTP колачиће - неки од параметара су тајни, а неке задаје корисник.

Отворени текст пре компресије изгледа овако:

```
password=tajna&theme=dark
```

Међутим, нападач може злоупотребити чињеницу да други параметар задаје корисник и покушати да погоди први (тајни) параметар тако што ће мењати вредност другог параметра и пратити резултате компресије. Што се више фраза првог и другог параметра понавља, то ће величина компресованог текста бити мања, а најмања ће бити када отворени текст буде изгледао овако:

```
password=tajna&theme=tajna
```

Сваки пут када нападач погоди једно слово, величина колачића ће се смањити за 1 бајт. Ово ради само у случају да су тајни параметар и кориснички параметар компресовани заједно.

Да би овај напад био могућ, потребно је да нападач може да види одговор сервера (барем величину након компресије), као и да има контролу над неким подацима на страници. Нпр. у "човек у средини" сценарију, нападач на овај начин може украсти CSRF (енгл. *Cross-Site Request Forgery*) токен Веб апликације иако је HTTP комуникација шифрована (преко SSL протокола). Ово је основа за CRIME и BREACH нападе.

4.4.2 CRIME напад

CRIME (енгл. *Compression Ratio Info-leak Made Easy*) је напад који експлоатише поменути проблем са шифровањем и компресијом код SSL протокола. Према SSL/TLS спецификацији, клијент шаље листу подржаних алгоритама компресије у ClientHello поруци, сервер бира један од понуђених или без компресије и затим шаље податке о изабраном алгоритму у ServerHello поруци (технички, клијент мора увек да понуди "*no compression*" као опцију). Оно што је специфично за овај напад је и да је могућ ван коришћења SSL протокола, тачније при коришћењу експерименталног протокола SPDY компаније Гугл.

У већини случајева, Веб клијенти су и конфигурисани тако да не захтевају компресију од сервера - шта више једина два Веб прегледача која су и могла да подрже компресију су два најпопуларнија прегледача из последње деценије, Chrome и Firefox. На серверској страни је

такође битно искључити било какве директиве везане за компресовање. На примеру Апач веб сервера, директива коју користимо је:

```
SSLCompression off
```

Јединствени идентификатор за CRIME напад је CVE-2012-4929. Аутори CRIME напада су аргентинац Ђулијано Рицо (енгл. *Juliano Rizzo*) и вијетнамац Таји Дуонг (енгл. *Thai Duong*). Напад је представљен 2012. године на *Ekoarty* конференцији.

4.4.3 BREACH напад

BREACH (енгл. Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) напад функционише исто као и CRIME напад, само што не ради на компресији код SSL протокола већ на HTTP компресији - практично не сме бити искључена јер би перформансе драстично опале. То значи да заштита од CRIME напада не може да се примени и на BREACH, јер искључивање SSL компресије није везано за HTTP компресију. Као могуће решење, IETF је предложио увођење падинга који би сакрио оригиналну величину компресованих података, међутим драфт је истекао без стандардизације. Обећавајуће решење би била заштита од CSRF пропуста (главни вектор за BREACH нападе) у виду *Same-Site* атрибута за HTTP колачиће. Ипак, ово још увек није подржано у свим Веб прегледачима.



Слика 11. BREACH напад

4.4.4 Временски напади - TIME и HEIST

Док CRIME и BREACH раде само у случају сценарија "човек у средини", на *Blackhat Europe 2013* конференцији су представљене две варијације које не подразумевају задати сценарио. TIME¹⁸ (енгл. *Timing Info-leak Made Easy*) и HEIST¹⁹ (енгл. *HTTP Encrypted Information can be Stolen through TCP-windows*) су примери оваквих напада где нападач не мора преко мреже да ишчита величину компресованих података, већ преко Јаваскрипта мери време компресованих и шифрованих захтева. Нападач (само) треба да натера жртвин (клијентов) Веб прегледач да пошаље на стотине или хиљаде захтева и затим упореди време потребно за одговор сервера. Што је време мање, то је одговор краћи (мања величина након компресије).

18 A Perfect CRIME? Only TIME Will Tell, <https://www.blackhat.com/eu-13/briefings.html#Beery>

19 <https://www.blackhat.com/docs/us-16/materials/us-16-VanGoethem-HEIST-HTTP-Encrypted-Information-Can-Be-Stolen-Through-TCP-Windows-wp.pdf>

Рањивости на нападе који користе проблеме који настају услед коришћења компресије у SSL протоколу могу бити детектовани на бесплатном сервису компаније High-Tech Bridge на локацији: <https://www.htbridge.com/ssl>.

4.5 Криптографски пропусти

Способност да међусобно деле шифроване поруке или дешифрију поруке непријатеља може дати државама одлучујућу предност у рату - нпр. током II светског рата британске морнарице су често знале када су мета немачких па су имале времена да се припреме и избегну напад или позову појачање, све захваљујући чувеним криптоаналитичарима из Блечли парка (енгл. *Bletchley Park*). Због тога, влада Сједињених Америчких Држава је се дуго опирала да јавно објави математичке детаље јаким крипто алгоритама (као што је RSA) и класификовала алгоритме као оружје, тј. стављали су забрану криптографима из САД да деле алгоритме са колегама из других држава, а на употребу криптографије била је од стране закона гледано као и на употребу ватреног оружја. Криптографи из САД су лобирали да се забрана укине, што због напретка криптографије као науке, што због тога да би могли међусобно да деле шифроване поруке које њихова влада неће моћи да дешифрије, и током 2000. године већина ових забрана је укинута.

Током лобија, NSA (енгл. *National Security Agency*) је се залагала за то да забране остану, сматрајући да би свеprisутност шифровања угрозила њихове SIGINT и COMINT програме. Док је наизглед изгледало да су забране потпуно укинуте, случајно или не, крипто алгоритми увезени од САД у друге државе су временом испољавали озбиљне безбедносне пропусте.

4.5.1 FREAK напад

FREAK (енгл. *Factoring RSA Export Keys*) је пропуст пронађен у неколико SSL имплементација (између осталог и у OpenSSL имплементацији). Пропуст је откривен почетком 2015. године, међутим датира још из '90-их година и времена укидања забране на извожење крипто алгоритама. Пронашао га је тим састављен од безбедносних експерата из Microsoft Research центра, као и шпанског IMDEA, односно француског INRIA института. Јединствени идентификатори су додељени овим пропустима пронађеним у различитим SSL имплементацијама:

- CVE-2015-0204 - OpenSSL
- CVE-2015-1637 - Microsoft Security Support Provider Interface - SSPI - који користи Веб прегледач компаније Мајкрософт - Internet Explorer
- CVE-2015-1067 - имплементација коју користи Веб прегледач компаније Епл - Safari

Пропуст је такође пронађен и у подразумеваном Веб прегледачу на Андроид оперативном систему компаније Гугл. Занимљиво је и да су велики .gov сајтови у САД били рањиви на ово: fbi.gov, whitehouse.gov и nsa.gov.



Слика 12. Хакерски напад на CIA, Рендал Монро, *xkcd.com*.

Последице напада су биле катастрофалне, због тога што су ослабиле кључеве коришћене за потребе асиметричне криптографије довољно да могу бити разбијени лако, и то не само од стране већих државних организација. Кроз медије је откривање пропуста повезано са укидањем забране извоза на крипто алгоритме, међутим никакви чврсти докази нису пронађени. Ипак, јасно је да овако нешто даје јасну поруку да свака држава која мисли на своју безбедност треба да има сопствене крипто алгоритме.

И овај напад захтева сценарио "човек у средини". У тој ситуацији, нападач може да уназади (енгл. *downgrade*) конекцију тако да уместо стандардног RSA алгоритма користи тзв. EXPORT GRADE RSA алгоритам, са кључевима величине 512 бита[24]. У време '90-их, ово је гарантовало NSA да може да разбије сваки шифрат који користи тај кључ. Међутим, рачунарска снага је расла у складу са Муровим законом, тако да у XXI веку овај шифрат може да разбије свако са малим или никаквим улагањима.

Ипак, веровало се да EXPORT GRADE алгоритми не могу да нанесу штету у SSL конекцијама. Разлози за то су били што у теорији већина модерних клијената или сервера ни не подржава избор EXPORT GRADE алгоритама у фази преговора. У пракси, ово се показало другачијим. Сценарио напада изгледа овако:

1. У ClientHello поруци клијент захтева коришћење стандардног RSA алгоритма
2. Нападач пресреће поруку, и мења садржај у "Export RSA"
3. Сервер одговара са кључем величине 512 бита (порука је потписана са сервером оригиналним, јачим, кључем)
4. Клијент прихвата слаб кључ величине 512 бита - пропуст

5. Нападач факторише RSA модуло и долази до RSA кључа за дешифровање
6. Када клијент шифрује *Pre-Master Secret* параметар тим кључем, нападач може да дешифрује све и дође до *Master Secret* параметра
7. Сада нападач може да дешифрује сваку поруку између клијента и сервера, а додатно може и да се лажно представи у комуникацији као било која од те две стране

Да ствари буду још горе, како је генерисање новог RSA пара кључева рачунарски релативно захтевна операција, већина сервера не ради ово за сваку нову конекцију. Нпр. код Апач Веб сервера, модул *mod_ssl* подразумевано генерише један EXPORT GRADE RSA пар кључева када је сервер покренут, и све до поновног покретања користи један исти кључ за све конекције. То значи да нападач ако дође до тог кључа може да разбије сваки шифрат који сервер пошаље, све док може да се убаци као "човек у средини" у сесији између клијента и сервера.

Пропуст је отклоњен релативно лако, али је опоменуо све криптографе на потенцијалне бекдор (енгл. *backdoor*) проблеме, не само у самим протоколима, већ и у крипто алгоритмима које ти протоколи користе.

4.5.2 Logjam напад

Иста група научника која је открила FREAK пропуст представила је и Logjam напад 2015. године. Logjam такође користи пропуст за EXPORT GRADE крипто алгоритам, само што се за разлику од FREAK напада који се базира на RSA алгоритму за шифровање, Logjam базира на експлоатацији слабог Дифи-Хелман алгоритма за размену кључа. Конкретно, на кључеве до 1024 бита. Протоколи који су угрожени су HTTPS, SMTPS и IMAPS. Последице успешне експлоатације пропуста су исте као и код FREAK напада - нападач може да дешифрује поруке у комуникацији, као и да се представи као једна од две стране у комуникацији.

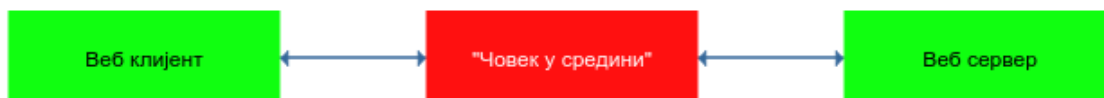
Дифи-Хелман алгоритам се базира на претпоставци да ће нападачу требати абнормално много времена да реши проблем дискретног логаритма. Аутори напада су се фокусирали на алгоритам сита поља општег броја (енгл. *General Number Field Sieve, GNFS*), који представља најефикаснији класични алгоритам за факторизацију великих целих бројева. GNFS алгоритам се састоји од четири велика корака, али прва три корака могу бити израчунати и кеширани, па се четврти корак сам може одрадити у релативно кратко временском периоду. Научницима је требало неколико хиљада CPU језгара да за недељу дана израчунају прва три корака за прост број величине 512 бита, али једном кад су прва три корака била кеширана четврти је могао бити одрађен за мање од минута користећи 2 Intel Xeon CPU са 18 језгара[25].

Препоруке за заштиту од оваквих напада су коришћење простих бројева величине 2048 бита код Дифи-Хелман алгоритма, као и ECDH алгоритма (енгл. *Elliptic Curve Diffie-Hellman*). Јединствени идентификатор овог пропуста је CVE-2015-4000.

Безбедност кључева сервера и његова ДН подешавања могу бити проверени на бесплатном сервису компаније High-Tech Bridge на локацији: <https://www.htbridge.com/ssl>.

5 Напад "Човек у средини"

У претходним секцијама прошли смо кроз најпознатије и најозбиљније нападе на SSL протокол и видели смо да већина њих подразумева да нападач буде "човек у средини" (енгл. *Man in the middle, MITM*). Као што је већ поменуто, сценарио "човек у средини" подразумева да нападач нарушава концепт поверљивости тако што надгледа комуникацију и додатно може да наруши интегритет комуникације тако што се представи у комуникацији као једна од нападнутих страна[26].



Слика 13. "Човек у средини" се понаша као малициозни прокси између Веб клијента и Веб сервера

Као безбедносни протокол, SSL води довољно рачуна да, уз правилну имплементацију, гарантује да су обе стране у комуникацији оне за којим се представљају. Међутим, напад "човек у средини" је могућ ако је се десио један од следећих услова:

- Приватни кључ сервера је украден - нападач може да се представи као сервер и клијент не види разлику између сервера и нападача
- Клијент верује небезбедном сертификационом телу - небезбедно сертификационо тело је сертификационо тело са малициозним намерама или валидно сертификационо тело где је корени приватни кључ украден - у том случају сертификационо тело може генерисати пар кључева за нападача који су потпуно идентични као и код сервера. Узевши у обзир да постоји гомила сертификационих тела данас, ово већ постаје озбиљан проблем. Овај сценарио укључује и могућност да је клијент нападнут, и у његову листу сертификационих тела којима верује додато и малициозно сертификационо тело.
- Клијент се не труди да верификује сертификат - нпр. пропуст у имплементацији клијента.

За нападача, постоје два типа "човек у средини" сценарија:

- пасиван - нападач само прислушкује, нарушује се концепт поверљивости
- активан - нападач учествује у комуникацији лажно се представљајући као једна од валидних страна - нарушава концепте поверљивости и интегритета

Осим у сврхе нарушавања приватности, сценарио "човек у средини" је користан и у области тестирања пробојности софтвера (енгл. *Penetration testing, pentesting*). Постоји више прокси алата који су намењени за ову сврху, тј. да се поставе као прокси између апликације или сервиса над којом се тестирање ради и самог тестера, тако да тестер може да дешифроване детаље из комуникације. Неки од тих алата су:

- Burp Suite²⁰, бесплатна или комерцијална верзија
- mitmproxy²¹
- OWASP Zed Attack Proxy - ZAP²²
- Fiddler²³, комерцијално решење

5.1 Подешавање HTTPS проксија на Burp Suite алату

Прављење HTTP проксија је релативно лако, може бити одрађено у неколико стотина линија кода у неком програмском језику високог нивоа користећи сокете. Међутим, данас се ретко срећемо са HTTP протоколом - сертификати се могу наћи бесплатно уз аутоматизоване скрипте које отклањају било какве компликације током инсталације, велике Интернет претраживачи попут Гугла такође приморавају Веб сајтове да користе HTTPS верзију, јер дају предност у претрази сајтовима који имају валидан SSL сертификат. Да би пентестер тестирао пробојност сајта који има SSL сертификат, потребно је да користи алат који као основну функционалност има HTTPS прокси.

Burp Suite је пример таквог алата. Постоји у комерцијалној и бесплатној верзији, али је бесплатна верзија сасвим довољна - комерцијална верзија садржи скенер за рупе у Веб апликацији (енгл. *Web vulnerability scanner*). Уколоко је ова функционалност баш потребна пентестеру може користити пројекат отвореног кода иза ког стоји OWASP организација - Zed Attack Proxy. У овом раду представимо конфигурацију Burp HTTPS проксија на два различита окружења: Мозила Фајрфокс Веб прегледач на Линукс Минт оперативном систему (тестирање Веб апликација), и Андроид оперативном систему (тестирање мобилних апликација).

5.2 Тестирање Веб апликација користећи Burp

За пример тестирања Веб апликација преко Burp алата користимо Мозила Фајрфокс Веб прегледач. Када покренемо Burp, прокси је аутоматски укључен. Прво што требамо да

20 <https://portswigger.net/burp>

21 <https://mitmproxy.org/>

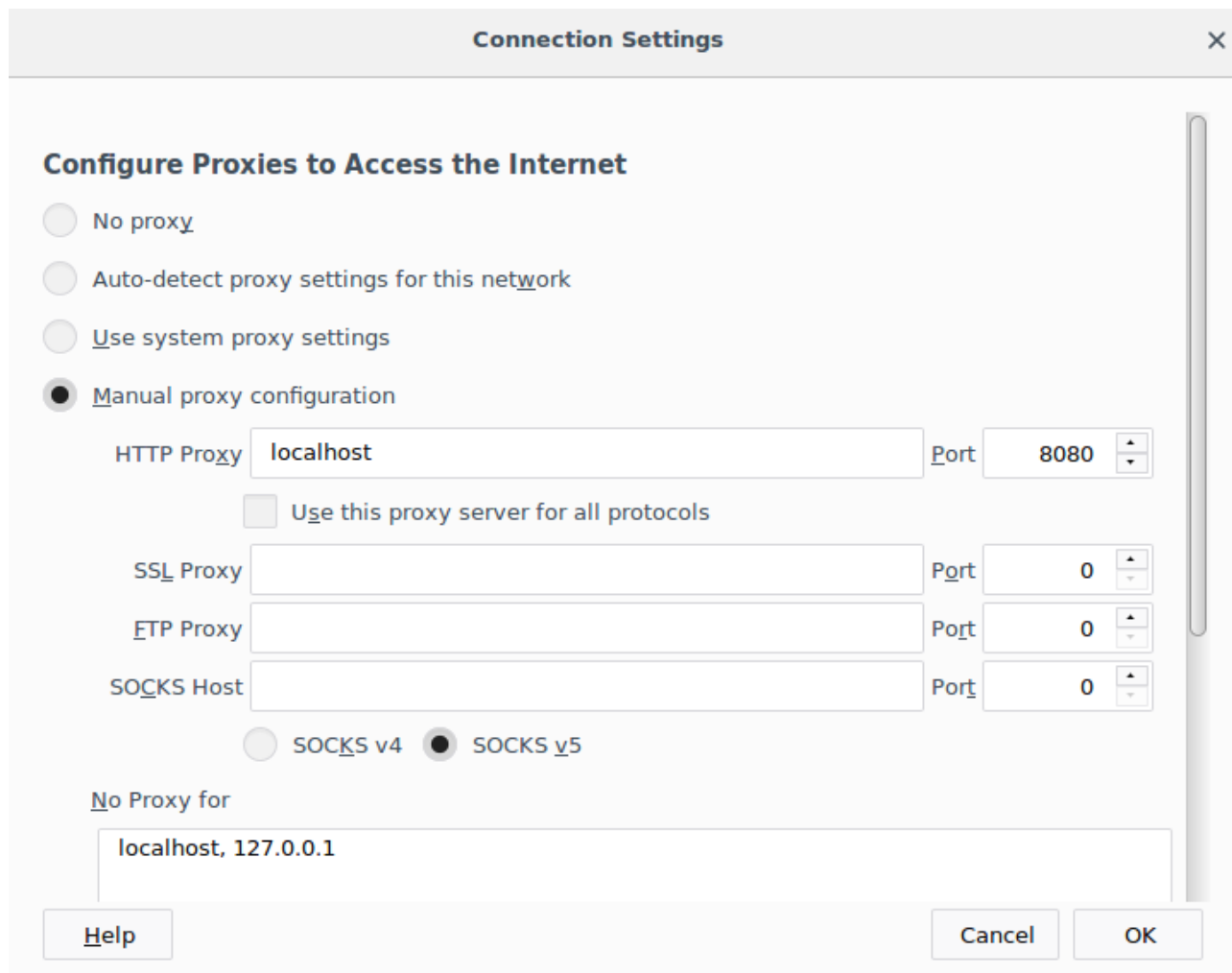
22 <https://github.com/zaproxy/zaproxy/>

23 <https://www.telerik.com/fiddler>

урадімо је да конфигуришемо наш Веб прегледач да шаље захтеве преко Вир проксија. На примеру Фајрфокс верзије 57, путања је следећа:

Settings > Preferences > General > Network Proxy > Settings

Изабраћемо *Manual proxy configuration*, са адресом localhost:8080 и чекираћемо *Use this proxy server for all protocols*. Евентуално ако тестирамо апликацију у локалу можемо уклонити localhost из поља *No Proxy for*.



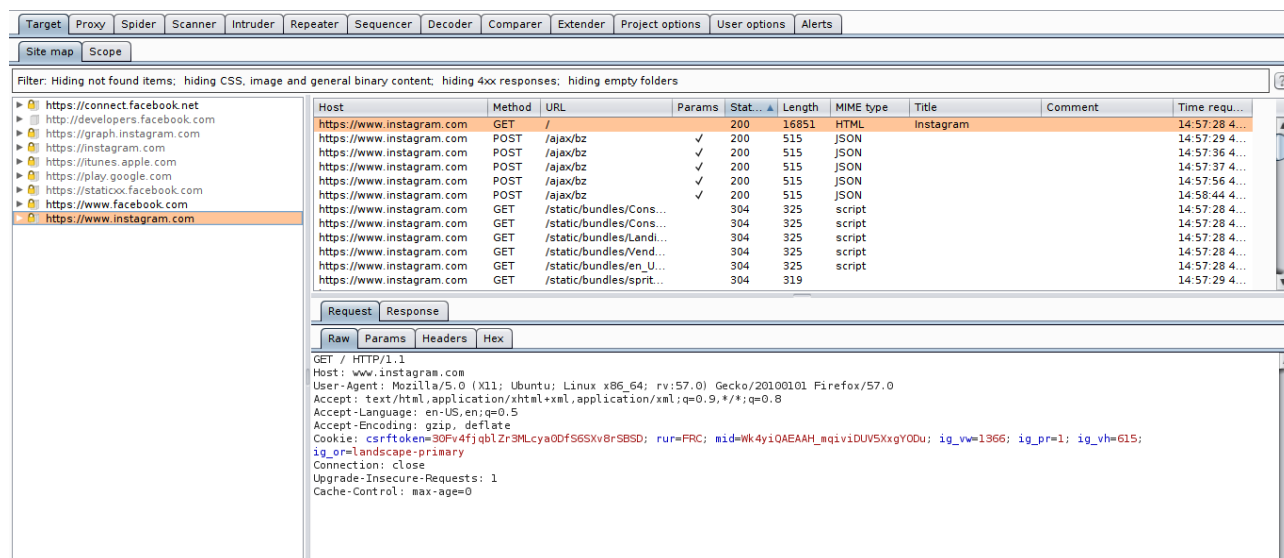
Слика 14. Подешавање Вир проксија у Мозила Фајрфокс прегледачу

Следећи корак је да отворимо <http://bitr> страницу у истом прегледачу и кликнемо дугме *CA Certificate* за генерисање сертификата који ћемо користити за HTTPS. Тај сертификат морамо да додамо у сертификате којима верујемо преко Фајрфоксовог менаџера сертификата. Менаџер сертификата можемо наћи на следећој путањи:

*Settings > Preferences > Privacy & Security > Certificates > View Certificates...
Authorities > Import # чекирамо Thrust this CA to identify websites*

Напомена: кад додамо Вур сертификат међу сертификате којима верујемо, морамо водити рачуна о том кључу као и да је кључ продуцкионог сервера у питању. У случају да нападач дође до тог кључа, може да се убаца као "човек у средини" у нашу SSL конекцију не приметно, чак и кад не користимо Вур прокси. Да би се заштитили од овог, Вур сваки пут генерише јединствени сертификат, који корисник складишти на фајл систему по избору - ако постоји опасност да нападач приступи садржају нашег фајл система онда боље не инсталирати Вур сертификат.

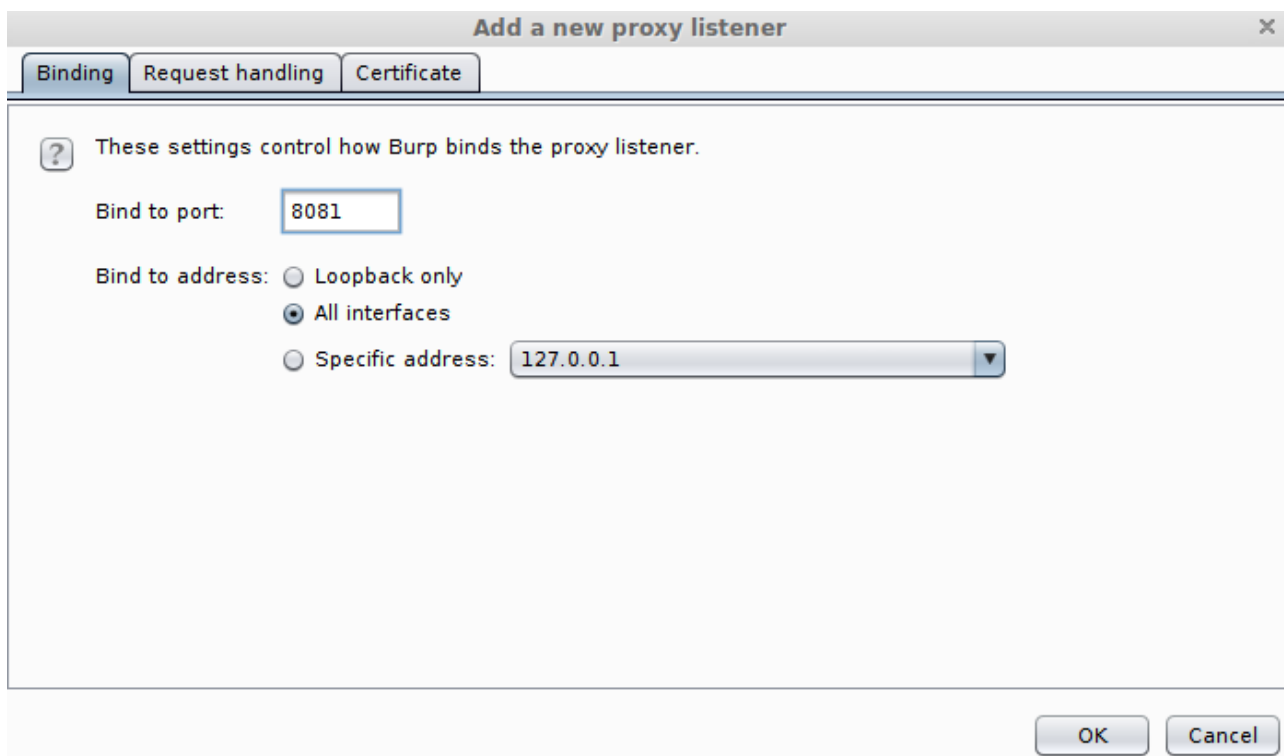
Ако смо правилно одрадили све наведене кораке изнад, сад би требали да имамо функционалан HTTPS прокси, као на слици испод.



Слика 15. Вур HTTPS прокси

5.3 Тестирање мобилних апликација

Тестирање мобилних апликација је другачије него тестирање Веб апликација. Веб апликације се извршавају унутар Веб прегледача. Кад подесимо прокси на самом Веб прегледачу, свака Веб апликација коју покренемо унутра користиће тај прокси. Мобилне апликације се међутим не извршавају унутар Веб прегледача, јер је на смартфонима Веб прегледач засебна мобилна апликација. Да би конфигурисали проксија мобилне апликације, морамо конфигурисати проксија на системском нивоу самог смартфона. Такође, као и код Веб прегледача, морамо додати Вур сертификат међу сертификате којима верујемо. Пошто Вур подразумевано ради само на тзв. loopback адреси - 127.0.0.1, а Вур на нашој машини и смартфон не деле исту мрежну картицу, морамо конфигурисати Вур проксија тако да прихвата конекције и ван локалне машине. Ово можемо постићи тако што ћемо на путањи *Proxy > Proxy Listeners* додати нови прокси интерфејс који ће радити на свим мрежним интерфејсима (у овом случају loopback и етернет), као што је приказано на слици испод.



Слика 16. Конфигурација Burp проксија тако да ради на свим мрежним интерфејсима

Смартфон мора бити у истој мрежи као и машина на којој се налази Burp прокси. На Андроид оперативном систему верзије 5.0 путања је следећа:

*Settings > Wi-Fi > *Network SSID* > Advanced Settings > Proxy*

Након конфигурације проксија на Андроиду, прво морамо подесити неку заштиту екрана (због полисе Андроида где не дозвољава инсталирање нових сертификата без тога). Нови сертификат додајемо преко следеће путање:

Settings > Security > Credential Storage > Install from SD card...

Некад се дешава да Burp не препозна уређај па генерише сертификат који не подржава Андроид оперативни систем (нпр. DER формат). Конверзију у подржани формат (PEM) можемо одрадiti преко OpenSSL алата, било на другој машини, било на Андроиду (рецимо преко Termux емулятора).

```
openssl x509 -inform der -in cacert.der -out burp.pem
```

Уколико је све одрађено како треба, сада можемо надгледати цео мрежни саобраћај на нашем Андроид телефону, чак и онај шифровани преко SSL протокола, какав и користе многе апликације.

Осим што нуди SSL MITM прокси функционалност, у својој бесплатној верзији Burp нуди још гомилу алата корисних за тестирање пробојности Веб апликација:

- "Уљез" (енгл. *Intruder*) - аутоматизовани напади на Веб апликације: уметање SQL кода, унакрсно скриптовање, манипулација параметрима у заглављу и сл.
- Спајдер (енгл. *Spider*) - имплементација crawler-a
- "Репетитор" (енгл. *Repeater*) - ручно тестирање апликације - модификација захтева (нпр. игнорисање заглавља, додавање колачића и сл.)
- Секвенцер (енгл. *Sequencer*) - анализа квалитета рандом параметара - нпр. корисно за тестирање CSRF токена и сличних параметара од којих се захтева одређен ниво случајности
- Декодер (енгл. *Decoder*) и компаратор (енгл. *Comparer*)

Burp такође дозвољава и проширење функционалности кроз инсталирање екстензија. На свом сајту нуди повећи избор екстензија на локацији: <https://portswigger.net/bappstore>.

Безбедност мобилних апликација је могуће тестирати преко бесплатног сервиса компаније High-Tech Bridge на локацији: <https://www.htbridge.com/mobile>. Сервис детектује рањивости преко статичке (енгл. *Static Application Security Testing - SAST*) и динамичке анализе (енгл. *Dynamic Application Security Testing - DAST*), као и анализу бекенд комуникације апликације, тј. са којим API тачкама тестирана апликација комуницира. Подржава све апликације са Гуглове Андроид продавнице, као и тестирање апликација за које поседујемо инсталациону архиву (APK за Андроид односно IPA за iOS).

6 Тестирање конфигурације SSL протокола

У претходном делу рада наведена су нека решења за тестирање SSL конфигурације на серверу. Сви ти алати или не раде парсовање одговора или га раде и даље на примитивном нивоу. Такође, врло често покривају само неке аспекте конфигурације а не све. Једно од решења за комплетно тестирање конфигурације SSL протокола је бесплатан сервис швајцарске компаније High-Tech Bridge доступан на адреси <https://www.htbridge.com/ssl>. Сервис се не ограничава само на Веб, већ је могуће тестирати и остале сервисе на Интернету (електронска пошта нпр.). Сервис је званично препоручен од стране OWASP пројекта (енгл. *Open Web Application Security Project*) и такође је део Мозилиног Observatory пројекта[13].

Такође, High-Tech Bridge нуди још један близак сервис на адреси <https://www.htbridge.com/radar>, где можемо тестирати опасност од напада типа cybersquatting, typosquatting и phishing за наше клијенте/посетиоце. Мале компаније немају адекватног начина да се одбране од оваквих напада, док велике компаније имају опцију једино да купују више домена. Овај проблем је отворио нову "браншу" у информационим технологијама - погађање домена (енгл. *Domain name speculation*). Најкупљиви домен продат у историји је (за сада) insurance.com, који је 2010. године купљен за 35,6 милиона долара. Проблемима са доменима се не види решење још од када је 2002. године америчка компанија Ford изгубила тужбу против најпознатијег хакерског часописа 2600 због линковања ка Фордовом Веб сајту са сајта на домену који садржи погрдне речи упућене Фордовом главном конкуренту у производњи аутомобила General Motors²⁴.

Cybersquatting - нападач прави Веб сајт на домену који је веома сличан домену жртве. Овре је жртва таргетирана унапред и линк ка лажном Веб сајту јој мора бити прослеђен на неки начин. Пример: google - google.

Typosquatting - нападач прави Веб сајт на домену који је веома сличан домену жртве, али за разлику од претходног случаја овде се нападач ослања на људску природу и очекује грешку у куцању (енгл. typo). Овде жртва није таргетирана директно, већ се нападач узда у необученост или непажњу корисника. Пример: google - gogle.

Phishing - нападач замаскира поруку тако да жртва верује да је аутор поруке прави пошиљаоц. Слободно се може рећи да је овај напад нека врста еволуираног социјалног инжињеринга имплементирана у виртуалном свету[27]. Најпознатији пример фишинга је свакако "Нигеријски принц", где се нападач жртви представља као члан краљевске породице неке земље трећег света (најчешће принц од Нигерије) и тражи почетни капитал уз обећање да ће се жртви вишеструко вратити новчано. Наравно, у случају да жртва наследи и уплати новац, нападач се више не би јавио.

24 http://www.theregister.co.uk/2002/06/29/ford_loses_2600_lawsuit/

Стандард за безбедност података у индустрији платних картица²⁵ (енгл. *Payment Card Industry Data Security Standard - PCI DSS*) је безбедносни стандард који морају да испуне Веб сајтови који нуде корисницима могућност плаћања картицом. Настао је 2004. године спајањем већ постојећих стандарда компанија Visa, MasterCard, American Express, Discover и JCB.

Закон о преносу здравственог осигурања и одговорности из 1996. године²⁶ (енгл. *Health Insurance Portability and Accountability Act of 1996 - HIPAA*) обавезује здравствене организације да заштићене информације о здрављу (енгл. *Protected health information - PHI*) не смеју бити доступне неауторизованим индивидуама.

Национални институт за стандарде и технологију²⁷ (енгл. *National Institute of Standards and Technology - NIST*) је главна метролошка установа у Сједињеним Америчким Државама основана 1901. године. Након сваке нове верзије SSL/TLS протокола објављује смернице за правилну конфигурацију сервера.

SSL сервис тестира имплементацију SSL протокола на задатом серверу у складу званичних публикација ове 3 установе.



SSL/TLS Server Test of facebook.com

Test SSL/TLS implementation of any service on any port for compliance with PCI DSS requirements, HIPAA guidance and NIST guidelines.



Слика 17. Бесплатна анализа SSL/TLS сервера

Неке од ствари које се анализирају при SSL/TLS тесту:

- Валидност и веродостојност сертификата
- Верзије протокола подржане од стране сервера

25 https://www.pcisecuritystandards.org/document_library?category=pcidss&document=pci_dss

26 <https://www.hhs.gov/hipaa/for-professionals/breach-notification/guidance/index.html>

27 <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf>

- Шифарски пакети подржани од стране сервера
- Подржавање савршене тајности унапред
- Подржавање крипто алгоритама базираних на елиптичкој криви (енгл. *Elliptic curve cryptography*)
- Рањивост на популарне нападе на SSL протокол (POODLE over TLS, Padding Oracle flaw, Heartbleed, OpenSSL CCS flaw и сл.)

Иста компанија нуди такође бесплатан сервис за тестирање HTTP заглавља - websec. Занимљиво, на websec сервису тестови бележе најслабије резултате. Као један од разлога за то намеће се чињеница да многа од препоручених заглавља нису ни индустријски ни *de facto* стандарди, већ само препоруке. Примера ради, тестира се да ли се не шаље Server заглавље, као и да ли постоје заглавља као Content Security Policy и већ поменути Strict Transport Security и Public Key Pins заглавља. Са друге стране имамо заглавља као што су X-XSS-Protection заглавље, које се користило пре мало пре поменутог CSP заглавља. Неко ко има добро конфигурисану CSP полису нема потребе за укључивањем X-XSS-Protection заглавља. Ипак, узевши у обзир да већина тестова нема ни CSP ни X-XSS-Protection заглавље, стиче се утисак да проблем ипак није до високих критеријума самог теста, већ и до конфигурације самог Веб сервера.



Слика 18. Бесплатна анализа HTTP заглавља

Већ поменути сервис radar служи да проверимо безбедност домена на наведене нападе. Додатно, сервис детектује налоге на друштвеним мрежама који садрже варијацију домена - уколико тај налог садржи и линк ка сајту тог домена, велика је вероватноћа да се ради о малициозном Веб сајту.

Potential Cybersquatting

DETAILS

Domains that look very similar to facebook.com domain, brand or business name.

Domains located in facebook.com subnetwork or with the same domain owner are not displayed below:

	Domain	Location / Server IP	Domain Registrar	Created
1	facebook-live.com	 199.59.242.150	NAMESILO, LLC	13.06.2017
2	facebooksolutions.com	 199.59.242.150	NAME.COM, INC.	21.05.2017
3	facebook-video.com	 185.24.98	REGISTER.IT SPA	15.05.2017
4	xn--faebook-n5a.com	 50.63.202.57	GODADDY.COM, LLC	14.05.2017
5	xn--facebok-bmb.com	 50.63.202.60	GODADDY.COM, LLC	02.05.2017
6	facebooklegal.com	 184.168.221.50	GODADDY.COM, LLC	28.04.2017
7	facebookservices.com	 173.239.23.228	GODADDY.COM, LLC	28.04.2017

Слика 19. Бесплатна анализа доменског имена за могуће cybersquatting/typosquatting/phishing нападе

Популарна алтернатива за бесплатне сервисе ове компаније је SSL Labs америчке компаније Qualys, који додуше нуди мање функционалности - нпр. тренутно није могуће као улазни параметар задати адресу Интернет протокола, већ је потребно унети доменско име - ово може бити ограничавајуће јер сервис може бити заштићен SSL протоколом а да нема домен. Иако не нуди рестрикције у броју тестова у одређеном временском интервалу, програмски API за SSL Labs није бесплатан, док High-Tech Bridge нуди потпуно бесплатан API, уз малу напомену да је у случају јавног коришћења потребно референцирати њихов сервис.

Последњи сервис је задужен за тестирање мобилних апликација. Сервис исписује основне податке о апликацији (назив, верзија, идентификатор, име аутора, привилегије које захтева на систему), као и рањивости апликације на OWASP Mobile Top 10 пропусте. Ипак, најзанимљивија функционалност коју сервис пружа за овај рад је свакако прокси који исписује све комуникације из ове апликације ка Интернету - овде можемо приметити да ли апликација користи са неком сумњивом адресом, или у случају да је адреса валидна и потребна апликацији да ли се користи шифрована верзија HTTP протокола. Нпр. мобилна апликација за едитовање текста нема потребу да комуницира са више сервера - евентуално 1-2 у случају да апликација садржи рекламе, док апликације за комуникацију или апликација за мобилни банкинг мора да комуницира са серверима само преко HTTPS протокола. Ово је било актуелно у ближој прошлости, јер до скоро популарне апликације за ћаскање као што су Viber или WhatsApp нису уопште имали шифровану комуникацију (данас обе апликације користе исто шифровање које користи и позната Signal апликација).

Mobile App Scanner

MAIN HOW-TO ABOUT API FEEDBACK

Test security and privacy of your mobile application (iOS & Android), detect OWASP Mobile Top 10 and other weaknesses:

 Upload your mobile app

 or

 Type Google Play app name

10 tests running 65.99K apps tested

☐ Hide from Latest Tests ☐ Debug 

Provided "as is" without any warranty of any kind

Mobile Application Audit Summary

	APP NAME	APP ID	APP VERSION	 PDF REPORT
	PIVAA	com.htbridge.pivaa	1.0	
	DEVICE TYPE	TEST STARTED	TEST FINISHED	 REMOVE TEST
Android	08.12.2017 22:22 CET	08.12.2017 22:28 CET		

Слика 20. Бесплатна анализа безбедности мобилне апликације (Android/iOS)

6.1 API за тестирање

Сервис такође нуди и апликативни програмски интерфејс (енгл. *Application Programming Interface* - *API*) који је бесплатан за коришћење са ограничењем од 250 захтева у року од 24 сата и максимално 20 захтева у року од 180 секунди са исте адресе Интернет протокола. У листингу испод дат је пример коришћења API позива преко cURL конзолног алата.

```
curl -d 'domain=twitter.com:443'
"https://www.htbridge.com/ssl/api/v1/check/$(date +%s).html"
```

Динамички параметар у униформном локатору ресурса који је унет унутар витичастих заграда (тзв. замена команди - енгл. *Command substitution*) се користи само због потреба забране кеширања на страни клијента (упит би радио да је последњи аргумент у наредби био <https://www.htbridge.com/ssl/api/v1/check/1.html>).

API је користан због могућности аутоматизације откривања пропуста у имплементацији SSL протокола у случају кад желимо да тестирамо више сервера, али и за сврхе прикупљања података који могу касније бити коришћени за неку даљу анализу или истраживање. Пример API позива користећи Python 3 је дат у листингу испод.

```

import requests
import time

data = {
    'domain' : 'twitter.com:443',
    'chosen_ip' : '104.244.42.193'
}
ts = str(int(time.time()))
url = 'https://www.htbridge.com/ssl/api/v1/check/' + ts + '.html'

r = requests.post(url, data=data)
if r.status_code == 200:
    data = r.json()
    # Оцена сервера на тесту
    print(data['results']['grade'])
    # Да ли је сервер испунио захтеве NIST-a
    print(data['nist']['compliant']['value'])
# end

```

Користећи програмски језик вишег нивоа (као што је у овом случају Python 3) можемо направити и напредније скрипте која би након прихватања резултата радила неку даљу анализу над добијеним резултатом.

7 Закључак

У оквиру првог дела рада представљен је начин функционисања рада SSL протокола и пример добре имплементације SSL протокола на Веб серверу. Истакнуто је да је утицај SSL протокола већи од заштите лозинке или података о кредитној картици. Јасно је да су те ствари најбитније за заштиту обичном човеку, али у данашњем свету, где је информација моћ, прикупљање толике количине података, макар они били наизглед небитни, може значити да разне треће стране на пасиван начин долазе до огромне количине информација, тј. моћи. Скренута је и пажња да SSL није панацеа за све проблеме на Интернету, већ само један од стубова Интернета сутрашњице - безбедног за све учеснике у комуникацији и са дозираним нивоом анонимности.

У другом делу рада представљени су различити типови напада на SSL протокол. Додатно, приказани су сервиси за детекцију рањивости које ови и слични напади експлоатишу. Треба имати у виду да ниједан сервис не детектује све могуће пропусте, као и да нису сви напади засновани на познатим пропустима. На први поглед чини се да је безбедност на Интернету поље где је нападач у великој предности због више различитих вектора напада. Међутим, није тако. Просечном кориснику је у већини случајева да буде сигуран довољно да редовно ажурира систем и програме (напади на системске пропусте и софтверске пропусте генерално), води рачуна при сурфовању Вебом и читању електронске поште (напади преко Интернета) и да не дозвољава физички приступ уређају особама којима не верује. Што се компанија тиче, ту је већ ситуација таква да нападач има много више вектора. Парадоксално, што је већа компанија, то је шанса нападача већа - треба имати на уму да је сваки ланац јак колико и његова најслабија карика, а у компанијама са великим бројем запослених нападачу игра закон великих бројева. Зато је компанијама битно да нон-стоп едукују своје запослене о безбедности, основним принципима заштите и смерницама како да безбедно користе основне Интернет сервисе. Добра литература за ово је књига једног од најпознатијих хакера свих времена, Кевина Митника (енгл. *Kevin Mitnick*) под насловом "Уметност обмане" (енгл. *The Art of Deception*). Књига се између осталог може наћи и на српском преводу, а садржи (између осталог) смернице за едукацију запослених о различитим векторима напада на дигиталне податке.

Литература

- [1] Младен Веиновић, Александар Јевремовић, Интернет технологије - Универзитет Сингидунум, 2013.
- [2] Michal Kosinska David Stillwella, Thore Graepel, Private traits and attributes are predictable from digital records of human behavior - The Psychometrics Centre, University of Cambridge, 2013.
- [3] RFC, Hypertext Transfer Protocol - HTTP/1.1 - 1999.
- [4] Александар Јевремовић, Младен Веиновић, Марко Шарац, Горан Шимић, Заштита у рачунарским мрежама - Универзитет Сингидунум, 2014.
- [5] Xueyang Xu, Z. Morley Mao, J. Alex Halderman, Internet Censorship in China: Where Does the Filtering Occur? - University of Michigan, 2011.
- [6] Младен Веиновић, Саша Адамовић, Криптологија 2 - Универзитет Сингидунум, 2013.
- [7] Mayra Rosario Fuentes, Cybercrime and Other Threats Faced by the Healthcare Industry - A TrendLabs Research Paper, 2017.
- [8], Security ROI: Fact or Fiction? - <https://www.csoonline.com/article/2123096/metrics-budgets/security-roi--fact-or-fiction-.html>, 2008.
- [9], How to calculate ROI and justify your cybersecurity budget - <https://www.csoonline.com/article/3010007/advanced-persistent-threats/how-to-calculate-roi-and-justify-your-cybersecurity-budget.html>, 2015.
- [10] RFC, The Transport Layer Security (TLS) Protocol - Version 1.2 - 2008.
- [11] Марко Шарац, Повећање поузданости SSL протокола применом вишеструких канала за аутентификацију - Универзитет Сингидунум, 2013.
- [12], Certbot and HTTP Public Key Pinning (HPKP) - <https://jkphl.is/articles/certbot-http-public-key-pinning-hpkp/>, 2017.
- [13], Transport Layer Protection Cheat Sheet - https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet, 2017.
- [14] Apache HTTP Server Version 2.4, Documentation - 2017.
- [15] Bruce Schneier, Applied Cryptography - John Wiley & Sons, 1996.
- [16] Александар Јевремовић, Интеграција сопствених криптолошких система у стандардну рачунарско-телекомуникациону инфраструктуру - Универзитет Сингидунум, 2011.
- [17] RFC, MIME Security with OpenPGP - 2001.
- [18] Christian August Holm Hansen, Analysis of Client Anonymity in the Tor Network - Norwegian University of Science and Technology, 2015.
- [19] RFC, Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS) - 2015.
- [20] Ioannis Lioupras, Eleni Manthou, Don't let my Heart bleed: An event study methodology in Heartbleed vulnerability case - Umeå University, 2014.
- [21], Padding oracle attack - https://en.wikipedia.org/wiki/Padding_oracle_attack, 2017.

- [22] Nadhem J. Al Fardan, Kenneth G. Paterson, Lucky Thirteen: Breaking the TLS and DTLS Record Protocols - 2013 IEEE Symposium on Security and Privacy, 2013.
- [23], Side-channel attack - https://en.wikipedia.org/wiki/Side-channel_attack, 2017.
- [24] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P. Strub, J. K. Zinzindohoue, A Messy State of the Union: Taming the Composite State Machines of TLS - IEEE Symposium on Security and Privacy, 2015.
- [25] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. Vander-Sloot, E. Wustrow, S. Zanella-Béguelink, P. Zimmermann, Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice - Microsoft Research, 2015.
- [26], Man-in-the-middle attack - https://en.wikipedia.org/wiki/Man-in-the-middle_attack, 2017.
- [27] Душан Панић, Игор Франц, Употреба информационих технологија у сврхе корпоративне шпијунаже - Међународни институт за безбедност, 2014.