|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Цвијетин Млађеновић, PR55-2017

Богдан Кондић, PR58-2017

**Демонстрација безбедносне комуникације и *Trojan Horse* напада**

Сигурност и безбедност електроенергетског софтвера

- Примењено софтверско инжењерство (ОАС) -

Нови Сад, 18.01.2021.

САДРЖАЈ

1. ОПИС РЕШАВАНОГ ПРОБЛЕМА
2. ОПИС КОРИШЋЕНИХ ТЕХНОЛОГИЈА И АЛАТА
3. ОПИС РЕШЕЊА ПРОБЛЕМА

# ПРЕДЛОЗИ ЗА ДАЉА УСАВРШАВАЊА

# ЛИТЕРАТУРА

# Опис решаваног проблема

Безбедност информација, понекад скраћена на infosec, јесте пракса заштите информација ублажавањем информационих ризика. То је део информационог управљања ризиком. Тиме је обично обухваћено спречавање или барем смањење вероватноће неовлашћеног/неприкладног приступа, употребе, откривања, ометања, брисања/уништавања, корупције, модификације, инспекције, евидентирања или девалвације, мада то може укључивати и смањење штетних утицаја инцидената.

Информације могу попримити било који облик, нпр. Електронски или физички, опипљиви (нпр. папирни документи) или нематеријални (нпр. знање). Основни фокус информационе безбедности је на балансираној заштити поверљивости, интегритета и доступности података, уз истовремено одржавање фокуса на ефикасној имплементацији смерница, без ометања продуктивности организације [1].

Сајбер-сигурност је пракса заштите мрежа, система и програма од дигиталних напада. Многи програмски језици се користе за обављање свакодневних задатака повезаних са сајбер-сигурношћу, али један од њих се појавио као индустријски стандард: *Python*, који доминира индустријом сајбер-сигурности.

*Python* има синтаксу која се лако чита и разуме и широк спектар апликација које га чине врло свестраним програмским језиком за било којег амбициозног професионалца у области сајбер-сигурности.Његов раст током последњих неколико година био је невероватан и сада се сматра једним од најпопуларнијих језика у свим индустријама.

*Python* се широко користи у безбедносном раду због своје синтаксе која се лако научи и широког спектра библиотека, које му пружају пуно функционалности. Иако се за извршавање ових задатака могу користити други језици, већина индустрије користи *Python* [2].

Проблем који се решава у овом раду састоји се из следећих целина:

* Клијент-сервер комуникација употребом *socket*-а
* Ослушкивање комуникације и крађа пакета
* Разбијање шифара
* Криптографија
* Коришћење *Trojan Horse* сајбер напада за прикупљање информација за *IP* адресу

*Socket* је библиотека мрежног повезивања на ниском нивоу која вам омогућава успостављање веза клијент-сервер. У контексту сајбер-сигурности, ово је важно јер вам омогућује повезивање са било којим уређајем на одређеном порту, са одређеним протоколом, као и слање података тој машини [2].

На основу различитог нивоа сигурности, у овом раду реализована су три различита сценарија:

1. Несигурна комуникација
2. Комуникација са претходним хешовањем лозинке пре њеног слања на сервер
3. Криптована комуникација коришћењем *AES* крипто алгоритма након размене симетричних кључева претходно криптованих употребом *RSA* крипто-алгоритма

# Опис коришћених технологија и алата

Технологије и алати који су коришћени приликом решавања овог проблема су следећи: *VMWare*, *VirtualBox*, *Kali*, *Ubuntu*, *Wireshark*, *dSniff*.

*VMware* је софтвер за виртуализацију на радним станицама. Покреће на *Windows*, *Linux*, и *Mac* *OS* *X* платформи, док се за велика предузећа користи хипервизор *VMware ESX* и *VMware ESXi* који се инсталира директно на сам сервер без претходно инсталираног било каквог оперативног система [3].

*VirtualBox* је веома моћан софтвер за виртуелизацију 32-битних и 64-битних оперативних система, на рачунарима са *Intel* или *AMD* процесорима. *VirtualBox* је једино професионално решење које је слободно доступно као *Open Source Software* под условима "*GNU*" верзије 2. *VirtualBox* ради на: *Windows*, *Linux*, *Mac* и *Solaris* оперативним системима [4].

*Kali Linux* је *Debian-based Linux* дистрибуција која има за циљ напредни тест за проверу пенетрације и ревизију сигурности. *Kali* садржи неколико стотина алата који су усмерени на различите задатке сигурности информација, као што су испитивање пенетрације, истраживање сигурнсоти, рачунарска форензика, инжењерство... *Kali Linux* је објављен 13. Марта 2013. године као потпуна реконструкција *BackTrack Linux*-а од врха до дна, која се у потпуности придржава *Debian* развојних стандарда. Укључено је више од 600 алата за тестирање пенетрације. Након прегледа сваког алата који је укључен у *BackTrack*, елиминисан је велики број алата који једноставно нису радили или су дуплирали друге алате који су вршили исту или сличну функцију. Детаљи о садржају налазе се на сајту *Kali Tools* [5].

*Ubuntu* је дистрибуција *GNU/Linux* оперативног система. Заснован на дистрибуцији *Debian*-а, *Ubuntu* највише пажње посвећује лакоћи коришћења, слободи од свих ограничења, редовном објављивању нових издања (сваких 6 месеци) и лакоћи инсталације. *Ubuntu* спонзорише приватно предузеће јужноафричког бизнисмена Марка Шатлворта, Каноникал. Назив дистрибуције потиче од афричке идеологије *ubuntu* (“хуманост према другима”), мада су предложена и друга значења [6].

*Wireshark* је бесплатан пакет анализатор, отвореног кода. Користи се за мрежно решавање проблема, анализу, развој софтвера, комуникационог протокола и образовање. Првобитно назван *Ethernal*, пројекат је преименован у *Wireshark* у мају 2006. године због проблема са заштитним знаком. *Wireshark* је укрштена платформа, користећи *GTK+* *widget toolkit* у текућим издањима, и *Qt* у развојним верзијама, да спроводе свој кориснички интерфејс и користећи *pcap* за снимање пакета. Ради на *Linux*, *OS X*, *BSD*, *Solaris*, као и неким другим оперативним системима налик на *Unix* и *Microsoft Windows*-у. Ту је и, заснована на терминалу, верзија звана *TShark*. *Wireshark* и остали програми дистрибуирани са њим као што је *TShark*, су бесплатни софтвери, објављени под условима *GNU General Public Licence* [7].

*dSniff* је скуп алата за њушкање лозинки и анализу мрежног промета које је написао истраживач безбедности и оснивач покрета Даг Сонг за рашчлањивање различитих апликационих протокола као и издвајање релевантних информација. *dSniff*, *filesnarf*, *mailsnarf*, *msgsnarf*, *urlsnarf*, и *webspy* пасивно надгледају мрежу у потрази за занимљивим подацима (лозинке, *e-mail*, датотеке, итд.). *Arpspoof*, *dnsspoof* и *macof* олакшавају пресретање мрежног промета који је нападачу обично недоступан. *Sshmitm* и *webmitm* имплементирају активне *Man-in-the-middle* нападе против преусмерених *SSH* и *HTTPS* сесија искоришћавањем слабих веза у *ad-hoc PKI* [8].

# Опис решења проблема

У овом поглављу описан је сценарио рада апликације.

Комуникација се врши између физички одвојених уређаја, који су повезани на исту локалну мрежу. Уређаји свој рад врше у посебним виртуелним окружењима, и разликујемо три кључне улоге у комуникацији:

1. Сервер
2. *Main-in-the-middle* нападач
3. Клијент

Сервер и клијент користе *Ubuntu* оперативни систем (слика 1), док *Man-in-the-middle* нападач користи *Kali* (слика 2). Иако су оба *Linux* оперативни системи и пружају велики број могућности, *Ubuntu* је више намењен за регуларне кориснике због саме лакоће његовог коришћења и целокупног искуства, док је *Kali* дистрибуција *Linux*-а управо намењена за кориснике који су способни да на напредан начин користе све сигурносне алате које им он пружа, те у овом случају више одговара за симулацију рада нападача него *Ubuntu*.

Сервер се са клијентима повезује употребом *socket* библиотеке. То представља елегантно решење јер омогућује мрежно повезивање на ниском нивоу са било којим уређајем на одређеном порту, са одређеним протоколом, као и слање података тој машини. Сервер у *main thread*-у константно ослушкује новопристигле конекције, након чега клијентима пружа опцију одабира сценарија њихове комуникације (несигурна комуникација, размена симетричног кључа за *AES* алгоритам или размена симетричног кључа за *AES* алгоритам, преходно криптованог употребом *RSA* алгоритма). Након одабира опције, за сваког појединачног повезаног клијента креира се посебан *thread* за комуникацију са истим (слика 3).

Пре почетка размене порука, у сваком сценарију рада апликације, сервер врши аутентификацију клијента, тј. тражи од њега да му пошаље креденцијале како би се успешно улоговао на систем. Након што му клијент проследи исте, сервер их чува у *LoginAttempts.txt* датотеци, намењеној за чување сваког покушаја пријаве на систем, затим проверава да ли су примљени *username* и *password* валидни, тј. да ли на систему постоји регистровани корисник којем они одговарају. Подаци о регистрованим корисницима чувају се у *RegisteredUsers.txt* датотеци (слика 4).

Комуникација између сервера и клијента несметано би се одвијала да не постоји могућност појаве нападача (*Man-in-the-middle*) има могућност да на разне начине саботира њихову раземну података. У овом истраживању, нападач користи такозвану *ARP spoofing* технику (слика 5).

У рачунарском умрежавању, *ARP spoofing*, *ARP cache poisoning*, или *ARP poison routing* јесте техника којом нападач шаље (лажиране) *Address Resolution Protocol* (*ARP*) порука на локалну мрежу. Генерално, циљ је повезати *MAC* адресу нападача са *IP* адресом другог *host*-а, као *default gateway*, што доводи до тога да се сав саобраћај намењен тој *IP* адреси уместо тога пошаље нападачу. *ARP spoofing* може омогућити нападачу да пресретне оквире података на мрежи, модификује или заустави саобраћај. Често се напад користи као отвор за друге нападе, као што су *denial of service* (*DOS*), *man in the middle*, or *session hijacking*. Овај напад може се користити само на мрежама које користе *ARP* протокол и захтева да нападач има директан приступ сегменту локалне мреже да би био нападнут [9].

Први сценарио рада апликације представља веома несигурну опцију, из разлога што се комуникација између сервера и клијента одвија без употребе *hash* и крипто алгоритама. Клијент кроз мрежу шаље своју лозинку серверу у “*plain text*” формату. Када ти подаци дођу до нападача, он без проблема да анализира садржај ухваћених пакета и врло лако долази до лозинке корисника, те овај режим комуникације није прихватљив.

У другом сценарију комуникације сервер и клијент приликом слања свог корисничког имена и лозинке претходно врши тзв. “хеширање” лозинке. Сигурносни алгоритам хеширања који се користи за добијање хеш вредности лозинке јесте *SHA256*.

*SHA-256* је сет криптографских хеш функција дизајниран од стране Националне Безбедносне Агенције (*NSA*). Криптографске хеш функције су математичке операције које се врше над дигиталним подацима; упоређивањем обрађеног "хеша" (резултата извршавања алгоритма) са познатом и очекиваном хеш вредношћу, можемо утврдити интегритет података. На пример, обрађивањем хеша скинутог фајла и упоређивањем резултата са претходно објављеном хеш вредношћу можемо утврдити да ли је фајл мењан или компромитован. Кључан аспекат криптографске хеш функције је његова отпорност на сударање, тј. није могуће наћи два различита уноса који као резултат дају исту хеш вредност [10]. *SHA256* је, како му и име говори, дужине 256 бита. Будући да *SHA256* враћа хексадецимални приказ, довољна су 4 бита за кодирање сваког знака (уместо 8, као за *ASCII*), па би 256 битова представљало 64 хексадецимална знака и његова дужина је увек иста, нимало не варирајући [11].

На овај начин, нападач приликом хватања пакета који садржи клијентову лозинку, уместо саме лозинке, он долази до њене хеширане вредности. Пошто су *hash* функције једносмерне и не постоји начин да се оне “*dehash*”-ују, то нападачу отежава посао, па то чини овај сценарио сигурнији од првог (слика 7). Ипак, постоје бројни алгоритми за разбијање (*crack*-овање) лозинки. У овом истраживању, представљени су *brute-force*, *dictionary*, као и хибридни алгоритам сачињен од та два.

У рачунарству, *brute-force* претрага или исцрпљујућа претрага, такође позната и као “генериши и тестирај”, је општа техника решавања проблема која се састоји од систематичног набрајања свих могућих кандидата за решавање и проверу да ли сваки кандидат задовољава проблем.

Док је *brute-force* претрага једноставна за примену, увек ћe пронаћи решење ако постоји, његова цена сразмерна је броју кандидата решења - што у многим практичним проблемима претендује на врло брз раст како се величина проблема повећава. Дакле, *brute-force* претрага се обично користи када је величина проблема ограничена или када постоји специфична хеуристика проблема која може бити искоришћена да би се смањио скуп решења кандидата до величине погодне за руковање. Метода се такође користи када је једноставност примене важности од брзине.

То је случај, на почетном, у критичним апликацијама где свака грешка у алгортиму има веома озбиљне последице; или када користите рачунар да докажете математичку теорему. *Brute-force* претрага је такође корисна и као „основни“ метод код бенчмаркинга других алгоритама или метахеуристике. Заиста, *brute-force* претрага може се посматрати као најједноставнији метахеуристик. Бруте форце претраживање не треба да меша са бектрекингом, где велики скупови решења могу бити одбачени без експлицитног набрајања.

Да би извршио напад *brute-force* алгоритмом, нападач може користити алат за покушај сваке комбинације слова и бројева, очекујући да ће на крају погодити лозинку. Ако нападач зна да организација захтева посебне знакове у својој лозинци, алат може добити упутства да садржи слова, бројеве и симболе. Свака лозинка, без обзира колико је јака, рањива је на овај напад. Међутим, овај метод ће потрајати (године, ако је лозинка довољно дугачка).

Време потребно за разбијање кратке лозинке (попут четвороцифреног *PIN*-а) може бити мање од једног минута. Проширење на шест знакова могло би потрајати сат времена. Проширење на осам знакова, са словима и симболима, може потрајати данима. Имајте на уму да сваки нови знак експоненцијално повећава количину времена потребног за грубу силу да би открио лозинку. Дакле, јака, дугачка лозинка може потрајати недељама или месецима. Али, уз довољно рачунарске снаге и посебно посвећеног нападача, лозинка би на крају била откривена [12].

Код *dictionary* напада, нападач користи листу речи у нади да је корисничка лозинка често коришћена реч (или лозинка виђена на претходним веб локацијама). Напади на речнике су оптимални за лозинке које се заснивају на једноставној речи (нпр. „*cowboys*“ или „*longhorns*“). Пописи речи нису ограничени на енглеске речи; често укључују и уобичајене лозинке (нпр. „*password*“, „*letmein*“ или „*iloveyou*“ или „123456“). Али савремени системи ограничавају своје кориснике од таквих једноставних лозинки, захтевајући од корисника да смисле јаке лозинке које се надамо да не би наћи на списку речи.

Када бисмо извршили поређење ова два алгоритма, лако је закључити да *dictionary* алгоритам далеко брже проналази лозинку од *brute-force* алгоритма у већини случајева, али не гарантује сигурност у проналаску лозинке (постоји велика шанса да се лозинка неће пронаћи у датотеци), док *brute-force* теоретски, уз довољно времена, може разбити било коју лозинку (слика 8) [13].

Такође, поред ова два алгоритма, у овом раду представљен је и хибридни алгоритам. Хибридни алгоритам представља комбинацију ова два алгоритма. Његов сценарио рада је следећи:

1. Покушати дешифровање *dictionary* алгоритмом
2. Уколико је први корак неуспешан, покренути *brute-force* алгоритам

На први поглед, можда делује временски неефикасно користити оба алгоритма, али чињеница је да *dictionary* алгоритам, чак и ако не нађе лозинку, јако брзо се завршава и одузима занемарљиво мало времена при покушају (слика 9), те ово представља оптимално решење.

Трећи сценарио рада апликације представља далеко најсигурнију опцију. У овом сценарију, клијент и сервер размењују симетричан кључ, који ће користити за *AES* крипто алгоритам (слика 10).

*Advanced Encryption Standard* (*AES*), познат и по свом оригиналном имену *Rijndael* је спецификација за енкрипцију електронских података основана од Националног института за стандарде и технологију у 2001. години. *Rijndael* је породица шифри са различитим величинама кључева и блокова. За *AES,* одабрана су три члана породице *Rijndael*, сваки са величином блока од 128 битова, али три различите дужине кључа: 128, 192 и 256 бита. *AES* је усвојила америчка влада и сада се користи широм света. Заснива се на принципу дизајна познат као мрежа замене-пермутације и ефикасан је и у софтверу и у хардверу. За разлику од свог претходника *DES*, *AES* не користи *Feistel* мрежу. *AES* оперише са низом бајтова великог реда величине 4 × 4 , названим *state*. Већина израчунавања *AES* алгоритма врши се у одређеном коначном пољу. Величина кључа која се користи за *AES* шифру одређује број рунди трансформације који претварају улаз, који се назива отворени текст, у коначни излаз, који се назива шифрирани текст. Свака рунда се састоји од неколико корака обраде, укључујући онај који зависи од самог кључа за шифровање. Скуп обрнутих рунди се примењује за трансформисање шифрованог текста натраг у оригинални отворени текст користећи исти кључ за шифровање [14].

Овај сценарио представља далеко сигурнију варијанту неко први сценарио, јер се врши енкрипција порука, које нападач не може да декриптује без симетричног кључа. Ипак, нападач се може инфилтрирати у комуникацију и пре саме размене кључа, и на тај начин ухватити пакет у ком је послат исти. Након што украде симетрични кључ, нападач има могућност декриптовања свог садржаја који клијент и сервер размењују.

Да би се то избегло, нужно је да клијент и сервер размене симетричан кључ на неки сигуран начин, где нападач не може да га користи. То ће бити реализовано употребом још једног крипто алгоритма – *RSA* (слика 11).

*RSA* је алгоритам за асиметричну криптографију, првенствено намењен шифровању података али се данас користи и у системима електронског потписа. *RSA* данас представља индустријски стандард у области асиметичне криптографије и заштити података, тако да је широко примењен у многим сигурносним протоколима и системи електронског пословања. У овом алгоритму кључну улогу имају велики прости бројеви. Сигурност *RSA* заснива се на сложености факторизације великих бројева. Сматра се да је одређивање оригиналне поруке на основу шифрата и кључа за шифровање еквивалентно факторизацији производа два велика проста броја [15].

Клијент и сервер претходно размене своје јавне кључеве, након чега клијент генерише симетричан кључ за *AES*, па помоћу серверовог јавног кључа употребом *RSA* алгоритма врши енкрипцију генерисаног симетричног кључа и тај садржај прослеђује серверу. Сервер ће пристигли криптовани кључ успешно декриптовати употребом свог приватног кључа.

На овај начин, нападач је немоћан да изврши декрипцију кључа, јер се садржај криптован серверовим јавним кључем може декриптовати једино употребом серверовог приватног кључа, ком он нема приступ.

Последња компонента система јесте вирус звани Тројански коњ (*Trojan Horse*).

Тројанским коњем се у рачунарском жаргону означавају штетни програм, који су „маскирани“ као корисни или се проширују „прикачени“ на друге корисне програме. „Тројанци“ обично врше, нежељене акције у рачунару и то у „позадини“ и прикривено. Најчешћа од тих нежељених акција је откривање корисничких лозинки, банковних података и других повјерљивих информација „прислушкивањем“ размјене података или једноставно читањем тих датотека, и јављање „власнику“ тројанског коња.

Постоје и тројански коњи у служби полиције који се баве прикупљањем информација са циљем откривања кривичног дела (*Remote Forensic Software*). Тај облик шпијунирања грађана је у неким земљама правоснажан и врши се по судском налогу (нпр. САД,Аустралија), у некима је и поред сукоба са Уставом, у фази припреме (Немачка, Аустрија, Швајцарска), док је у некима одбијен. Такви тројанци се шире инсталацијом или актуелизовањем комерцијалних оперативних система и других софтверских и хардверских компоненти рачунара, као и путем интернет провајдера инфилтрирањем у постојеће механизме преноса података, који такву могућност у својим продуктима и услугама, на захтјев дотичне државе, морају предвидјети [16].

У овом истраживању, тројански коњ се користи у виду игрице судоку. Корисник покреће игрицу, али поред тога, “у позадини” се врше малициозни процеси, који могу наштетити његовом рачунару. Нападач добија приступ терминалским командама док је игрица у току, што му омогућује приступ ресурсима тог рачунара (може да добије *IP* адресу, да промени шифру, да угаси рачунар итд.).

# Предлози за даља усавршавања

У овом поглављу нашег истраживања размотрићемо потенцијална побољшања која би довела до бољег и лакшег коришћења овог система. Са стране клијент-сервер архитектуре, потенцијално би се постојећа апликација могла проширити тако што би се омогућила регистрација нових кориника на систем тако што би нови корисник унео све своје потребне информације и на тај начин поднео захтев да буде корисник система. Након што би нови корисник поднео валидан захтев за регистрацију, систем би му одобрио захтев да постане корисник система.

Оваква имплементација регистрације нових корисника не би имала смисла док се пре тога систем не би проширио на начин да се јасно дефинишу типови корисника система тј. да се за сваког корисника веже његова улога. Да би се то постигло на целокупан систем би требало применити *rolebasedaccesscontrol*(*RBAC*) модел. На тај начин би у нашем систему имали различите типове корисника (нерегистрован корисник, регистрован корисник, админ итд.) и врло једноставно би могли да дефинишемо ком делу система који од типова корисника система може да се приступи.

Са стране нападача, могуће је увести проширење алгоритама за крековање лозинки. У проширеном систему, био би могућ одабир једносмерне функције (нпр. *SHA512*, *Shadow*, итд.) коју ће алгоритам користити приликом разбијања лозинки. На овај начин, нападач би врло лако могао да дође до оригиналне лозинке, која је прошла кроз било коју од тих функција.

# Литература

[1] ***https://sr.wikipedia.org/sr/Bezbednost\_informacija/***

[2]***https://simpleprogrammer.com/python-programming-for-cybersecurity/***

[3]***https://sr.wikipedia.org/sr-ec/VMware/***

[4]***https://sr.wikipedia.org/sr/VirtualBox/***

[5]***https://aleksandargavrilovic.wordpress.com/2018/03/01/sta-je-kali-linux/***

[6]***https://sr.wikipedia.org/sr/Ubuntu/***

[7]***https://en.wikipedia.org/wiki/Wireshark/***

[8]***https://en.wikipedia.org/wiki/DSniff/***

[9]***https://en.wikipedia.org/wiki/ARP\_spoofing/***

[10]***https://sr.wikipedia.org/sr-ec/SHA-2/***

[11]***https://stackoverflow.com/questions/2240973/how-long-is-the-sha256-hash/***

[12]***https://sr.wikipedia.org/sr/Iscrpna\_pretraga/***

[13]***https://www.rapid7.com/fundamentals/brute-force-and-dictionary-attacks/***

[14]***https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard/***

[15]***https://sr.wikipedia.org/sr-ec/RSA\_(алгоритам)/***

[16]***https://sr.wikipedia.org/sr-ec/Тројански\_коњ\_(информатика)/***