UDC: 004.738.2 INFO M: str. 16-24

METODE SIGURNOG PRENOSA PODATAKA IZMEĐU MOBILNE APLIKACIJE I UDALJENOG SERVERA
SECURE DATA TRANSMISSION METHODS BETWEEN MOBILE APPLICATION AND REMOTE SERVER

Danijela Zoran, doc. dr Zoran Đurić
Elektrotehnički fakultet Banjaluka, RS, BiH

REZIME: Sve je više mobilnih aplikacija koje rade sa osjetljivim podacima kao što su podaci o identitetu korisnika, finansijski podaci (npr. podaci o kreditnim karticama), povjerljivi poslovni podaci i sl. Kada se ti podaci prenose preko nesigurnog medija izloženi su raznim vrstama napada. Zbog toga je potrebno obezbijediti siguran prenos podataka. Postoje različiti sigurnosni mehanizmi na svakom od slojeva OSI referentnog modela. U ovom radu će biti analizirani različiti sigurnosni mehanizmi na transportnom i aplikacionom sloju (HTTP Basic autentikacija, HTTP Digest autentikacija, HTTPS i zaštita na nivou poruke). Neki od njih obezbjeđuju samo autentikaciju korisnika i ograničen pristup resursima na serveru, dok drugi obezbjeđuju sigurnu komunikaciju koja podrazumijeva tajnost, integritet podataka, kao i autentikaciju strana u komunikaciji. U radu su dati konkretni primjeri korišćenja pojedinih mehanizama na Android i iOS platformi.
KLJUČNE REČI: sigurnost, mobilne aplikacije, napadi, HTTP, HTTPS, Android, zaštita na nivou poruke
ABSTRACT: Number of mobile applications which manipulate sensitive data such as users's identity data, financial data (for example credit card data) and confidential business data, is growing every day. Data transmited over insecure medium is exposed to various types of attacks. Therefore it is necessary to ensure secure data transmission. There are different security mechanisms at each layer of the OSI reference model. This paper analyzes security mechanisms at application and transport layer (HTTP Basic authentication, HTTP Digest authentication, HTTPS and message level security). Some of these security mechanisms provide restricted access to servers's resources, while others provide secure communication that includes confidentiality, data integrity and authentication of communication parties. This paper contains several code examples, written for Android and iOS platform, with the sole purpose to illustrate the usage of mentioned security mechanisms.
KEY WORDS: security, mobile applications, attacks, HTTP, HTTPS, Android, message level security

1. UVOD

Sve je više mobilnih aplikacija koje rade sa osjetljivim podacima kao što su korisnički kredencijali, podaci o identitetu korisnika, finansijski podaci (npr. podaci o kreditnim karticama), povjerljivi poslovni podaci i sl. Često se ovakvi podaci šalju na obradu udaljenom serveru, preko nesigurnog medija kao što je Internet. Internet nije zamišljen da obezbijedi siguran prenos podataka, što znači da su podaci koji se prenose preko Interneta izloženi različitim vrstama napada. Ako zlonamjerni napadači dođu u posjed osjetljivih podataka mogu ih zloupotrijebiti. Zato je od izuzetne važnosti obezbijediti siguran prenos podataka između mobilne aplikacije i udaljenog servera.

U sekciji 2 dat je pregled osnovnih vrsta napada na sigurnost informacionih sistema i servisa u mrežnom okruženju. U sekciji 3 dat je kratak pregled savremenih kriptografskih mehanizama zaštite, uključujući tu simetrične i asimetrične algoritme, algoritme za kreiranje digitalnog sažetka, digitalni potpis, digitalnu kovertu i digitalni sertifikat. U sekciji 4 analizirane su metode za siguran pristup udaljenom serveru, i to: metode koje obezbjeđuju autentikaciju učesnika u komunikaciji na primjeru HTTP (*HyperText Transfer Protocol*) Basic i HTTP Digest autentikacije, metode koje obezbjeđuju zaštitu na transportnom sloju (na primjeru HTTPS protokola), kao i metode koje obezbjeđuju zaštitu na nivou poruke. Za svaku od navedenih metoda iznesene su osnovne karakteristike, prednosti i nedostaci, a dati su i odgovarajući primjeri njihove upotrebe na Android i iOS platformi.

2. VRSTE NAPADA

Iako se termini napad i prijetnja često poistovjećuju postoji suštinska razlika u njihovom značenju. Naime, prijetnja je mogućnost narušavanja sigurnosti sistema putem iskorišćavanja određenog propusta u samom sistemu, dok samo iskorišćavanje takve prijetnje predstavlja napad na sistem [1]. Napadi na sigurnost mogu biti pasivni i aktivni [2]. Pasivnim napadom napadač pokušava doći u posjed osjetljivih informacija (iz samog sistema, prisluškivanjem komunikacije između legitimnih učesnika i sl.), pri čemu ne utiče na sam sistem (npr. ne mijenjaju se resursi sistema, poruke u prenosu i sl.). Aktivnim napadom napadač pokušava doći u posjed osjetljivih informacija uticajem na sistem (npr. mijenjanjem poruka u prenosu, generisanjem novih i sl.) ili uticati na funkcionisanje sistema (npr. DoS - *Denial of Service* napadi i sl.).

2.1. *PASIVNI NAPADI*

Kod pasivnih napada cilj napadača je da dođe u posjed informacija koje se prenose između dva legitimna učesnika u komunikaciji. Prisluškivanje, kao najčešći tip pasivnog napada, odnosi se na presretanje poruka, obično bez detekcije tog

čina. Ovaj napad je posebno efikasan ako informacije koje se prenose nisu kriptovane, tj. ako se one prenose u otvorenom obliku. Međutim, čak i kada se informacije prenose u kriptovanom obliku to nije garancija da napadač ne može doći do korisnog sadržaja analizom saobraćaja u čiji je posjed došao prisluškivanjem[3, 4]. Analizom sadržaja moguće je uočiti pravilnosti u porukama i doći do informacija o identitetu i lokaciji pošiljaoca i primaoca, učestanosti slanja poruka, veličini poruka, ali i do sadržaja samih poruka.

Pasivne napade je teško otkriti jer se informacije koje se prenose ne modifikuju.

2.2. *AKTIVNI NAPADI*

Kod aktivnih napada napadač je u stanju da modifikuje podatke, kreira lažne, te ih šalje ka učesnicima u komunikaciji. Isto tako, napadač je u stanju da blokira komunikaciju između legitimnih učesnika. Neki od najčešćih aktivnih napada su [5]:

- lažno predstavljanje (eng. *Masquerade*),
- ponovno slanje poruka (eng. *Message replay*),
- MITM (*Man In the Middle*) i
- napad tipa ukidanja servisa (DoS, eng. *Denial of Service*).

Lažno predstavljanje podrazumijeva da se napadač predstavlja kao neki drugi legitimni entitet. To mu omogućava neautorizovani pristup podacima ili kreiranje lažnih podataka. Često se kombinuje sa drugim vrstama napada. Ponovno slanje poruka podrazumijeva da napadač presretne originalnu poruku i ponovo je pošalje, naknadno. Pri tome napadač ne mora poznavati sadržaj poruke. MITM napad je aktivni napad kod koga se napadač postavlja između dva legitimna učesnika u komunikaciji. Kod ovog napada napadač presreće poruke i modifikuje njihov sadržaj, predstavljajući se kao legitimni učesnik u komunikaciji. Napad tipa ukidanja servisa otežava ili u potpunosti onemogućava legitimnim korisnicima normalno korišćenje resursa sistema ili servisa. Ovakav napad se može izvršiti, npr. onesposobljavanjem kompletne komunikacione mreže ili preopterećenjem određenog servisa velikim brojem zahtjeva.

Aktivne napade je lakše otkriti, ali ih je teže spriječiti od pasivnih napada, zbog potencijalno mnogo većeg broja aktivnosti koje napadač može pokušati iskoristiti protiv sistema. Bitno je primijetiti da je određene aktivne napade praktično nemoguće spriječiti (npr. DoS napad sa udaljene mreže koja je u potpunosti pod kontrolom napadača). Iz tog razloga, često je cilj detektovati aktivni napad i poništiti njegove efekte, a ne spriječiti ga.

3. SAVREMENI KRIPTOGRAFSKI MEHANIZMI ZAŠTITE

U cilju rješavanja navedenih problema razvijaju se specijalizovani softverski i hardverski sistemi zaštite. Ovakvi sistemi su uglavnom bazirani na primjeni kriptografskih algoritama i tehnika. Da bi se jedan sistem mogao smatrati sigurnim, neophodno je da budu obezbjeđeni sljedeći sigurnosni zahtjevi [6]:

1. tajnost,
2. integritet,
3. neporecivost i
4. autentikacija.

Očuvanje tajnosti podrazumijeva da podaci budu dostupni samo legitimnim učesnicima u komunikaciji, koji su autorizovani da podacima i pristupaju. Integritet podataka se odnosi na činjenicu da primalac poruke mora biti u mogućnosti da verifikuje da poruka nije mijenjana u toku prenosa. Da bi se očuvao integritet potrebno je obezbijediti mehanizam za detektovanje neautorizovanog mijenjanja podataka. Obezbjeđivanje neporecivosti podrazumijeva da postoji mehanizam koji će onemogućiti učesnicima u komunikaciji da naknadno poreknu slanje poruka. Autentikacija se odnosi na utvrđivanje identiteta učesnika u komunikaciji.

Savremeni kriptografski algoritmi, koji se koriste za zaštitu informacionih sistema i servisa u mrežnom okruženju mogu se podijeliti na simetrične i asimetrične. Simetrični kriptografski algoritmi su algoritmi kod kojih se isti ključ koristi za enkripciju i dekripciju. Nazivaju se još i algoritmi sa tajnim ključem, jer je njihova sigurnost zasnovana na tajnosti ključa. Bitska dužina ključa direktno utiče na sigurnost ovih algoritama. Ključevi veće bitske dužine obezbjeđuju veći stepen zaštite. Ovi algoritmi se dijele na sekvencijalne i blok šifarske algoritme. Najpoznatiji sekvencijalni algoritam je RC4 (*Rivest Cipher 4*), a najznačajniji predstavnici blok šifarskih su DES (*Data Encryption Standard*), 3DES (*Triple Data Encryption Standard*), IDEA (*International Data Encryption Algorithm*) i AES (*Advanced Encryption Standard*). Osnovni nedostatak simetričnih kriptografskih algoritama je što se ključevi moraju distribuirati u tajnosti, što može biti težak zadatak. U slučaju kompromitacije ključa kompromitovane su sve poruke koje su enkriptovane datim algoritmom korišćenjem kompromitovanog ključa. Pod pretpostavkom da svaki par korisnika u mreži upotrebljava zaseban ključ za međusobnu komunikaciju, ukupan broj ključeva se brzo uvećava kako raste broj korisnika (broj ključeva = n*(n-1)/2, gdje je n broj učesnika u komunikaciji). Takođe, simetričnim algoritmima je nemoguće obezbijediti autentikaciju, integritet poruka i neporecivost slanja poruke. U nastojanju da se prevaziđu problemi vezani za simetrične algoritme, uvedeni su asimetrični algoritmi.

Kod asimetričnih algoritama za enkriptovanje i dekriptovanje se koriste različiti ključevi. Najznačajniji predstavnik asimetričnih algoritama je RSA (*Rivest Shamir Adleman*). Primjenom asimetričnih algoritama na odgovarajući način mogu se obezbijediti tajnost, integritet, neporecivost i autentikacija. Ipak, simetrični algoritmi na bazi slučajnog i pseudoslučajnog niza pružaju viši nivo sigurnosti od asimetričnih algoritama pri istim dužinama ključeva. Takođe, realizovanje asimetričnih algoritama je znatno zahtjevnije sa aspekata potrebnih resursa. Pored toga, asimetrični algoritmi su sporiji

od simetričnih. Stoga se, tipično, simetrični algoritmi koriste za očuvanje tajnosti podataka, a asimetrični za očuvanje integriteta, obezbjeđivanje neporecivosti i autentikaciju.

Za savremene kriptografske sisteme veoma su značajni i algoritmi za kreiranje digitalnog sažetka (eng. *message digest algorithms*). Ovi algortimi opisuju jednosmjernu transfromaciju poruke u jedinstveni izlaz određene bitske dužine koji se naziva digitalni sažetak ili heš (eng. *hash*) vrijednost. Najpoznatiji algoritmi ove vrste su MD5 (*Message Digest 5*), SHA-1 (*Secure Hash Algorithm 1*), SHA-224, SHA-256, SHA-384, SHA-512. Od navedenih algoritama, neadekvatni današnjim zahtjevima i preporukama smatraju se MD5 i SHA-1 algoritmi [7].

Integritet, neporecivost i autentikacija mogu se obezbijediti korišćenjem digitalnog potpisa. Digitalni potpis ne obezbjeđuje tajnost podataka. Da bi se digitalno potpisala neka poruka potrebno je primijeniti jedan od algoritama za kreiranje digitalnog sažetka, te izračunati heš vrijednost poruke. Zatim se heš vrijednost enkriptuje privatnim ključem pošiljaoca. Enkriptovana heš vrijednost se šalje zajedno sa porukom. Primalac dekriptuje poslatu heš vrijednost koristeći javni ključ pošiljaoca i poredi tu vrijednost sa heš vrijednošću koju izračuna na isti način kao i pošiljalac. Ako se te vrijednosti poklapaju primalac može biti siguran da poruka u prenosu nije mijenjana, kao i da je pošiljalac taj za koga se i predstavlja. Pored toga, pošiljalac ne može poreći da je poslao datu poruku.

Asimetrični algoritmi se mogu koristiti da se uz poruku enkriptovanu simetričnim algoritmom sigurno pošalje ključ simetričnog algoritma. Ključ simetričnog algoritma se enkriptuje javnim ključem primaoca i šalje se zajedno sa porukom koja je enkriptovana simetričnim algoritmom korišćenjem datog ključa simetričnog algoritma. Primalac svojim tajnim ključem dekriptuje ključ simetričnog algoritma, a onda pomoću tog dobijenog ključa dekriptuje primljenu poruku. Ova tehnika prenosa ključa simetričnog algoritma naziva se digitalna koverta. Za povezivanje identiteta učesnika u komunikaciji i njegovog javnog ključa koji se koristi u asimetričnim algoritmima koriste se digitalni sertifikati.

4. ANALIZA METODA ZA SIGURAN PRISTUP UDALJENOM SERVERU

4.1. *HTTP AUTENTIKACIJA*

Za razmjenu podataka između mobilne aplikacije i udaljenog servera najčešće se koristi HTTP protokol. Tako, na primjer, Android platforma omogućava da mobilna aplikacija sa udaljenim serverom komunicira putem HTTP protokola ili putem TCP *socket*-a [8]. Iako je dizajniran prvenstveno za prenos tekstualnih fajlova HTTP se može koristiti i za prenos multimedijalnih fajlova nakon što se oni konvertuju u tekstualni format. HTTP definiše niz standardnih pravila za uspostavljanje konekcije, autentikaciju, prezentaciju podataka pri prenosu preko Interneta i sl. [9]. HTTP je *stateless* protokol, što znači da ne omogućava praćenje korisničke sesije. Bitno je pomenuti da ovaj protokol nije dizajniran da podrži sigurnosne zahtjeve koji se postavljaju u savremenim računarskim mrežama. Osim pristupa samim podacima koji se prenose, napadači često nastoje promijeniti kolačiće i elemente HTTP zahtjeva i odgovora. HTTP obezbjeđuje jednostavan mehanizam za autentikaciju koji se zasniva na izazov-odgovor (eng. *challenge-response*) sistemu. Ako je pristup resursima na udaljenom serveru zaštićen, server zahtjeva od klijenta (mobilne aplikacije) da se autentikuje, nakon čega klijent šalje informacije potrebne za autentikaciju. Ako server primi zahtjev bez podataka potrebnih za autentikaciju on klijentu šalje HTTP odgovor sa statusnim kodom 401 (Unauthorized), što označava da klijent mora da se autentikuje. Ova HTTP poruka mora imati WWW-Authenticate zaglavlje čija se vrijednost sastoji od identifikatora autentikacione šeme koja će se koristiti za autentikaciju, nakon čega se navodi vrijednost *realm* polja, praćena sa prozvoljnim brojem parova atribut-vrijednost odvojenih zarezima koji su neophodni za autentikaciju korišćenjem date šeme. Autentikacioni parametar *realm* se navodi za sve autentikacione šeme. *Realm* vrijednost zajedno sa kanoničkim URL-om (*Uniform Resource Locator*) servera identifikuje zaštićeni prostor na serveru. Naime, resursi na serveru mogu pripadati različitim zaštićenim prostorima za koje se mogu koristiti različite autentikacione šeme, što je još jedan dodatni mehanizam zaštite. Klijent tipično odgovara porukom koja sadrži Authorization zaglavlje čija vrijednost su kredencijali (ostali podaci zavise od autentikacione šeme koja se koristi) koji su zahtijevani za autentikaciju. Ako su podaci validni korisnik ima pravo pristupa zaštićenim resursima. Isti kredencijali se mogu koristiti za sve zahtjeve u istom zaštićenom prostoru u periodu vremena koji određuje autentikaciona šema, parametri i/ili korisnikova podešavanja. Ukoliko server ne prihvata kredencijale primljene u zahtjevu, onda treba da vrati Unauthorized poruku. HTTP ne ograničava aplikacije samo na ovaj način kontrole pristupa već je moguće primijeniti i dodatne mehanizme zaštite kao što su npr. mehanizmi zaštite na transportnom ili mrežnom nivou.

4.1.1. HTTP BASIC AUTENTIKACIJA

HTTP Basic autentikacija [10] je najjednostavniji način kontrole pristupa, ali je ujedno i najmanje siguran. Naziv autentikacione šeme je Basic. Nakon što server primi neautorizovan zahtjev za resursom koji je u zaštićenom prostoru šalje klijentu odgovor koji sadrži WWW-Authenticate zaglavlje. Klijent šalje novi zahtjev sa zaglavljem Authorization, za čiju vrijednost specifikuje naziv autentikacione šeme praćene vrijednošću identifikatora korisnika i njegove lozinke odvojenih znakom „:“. Identifikator i lozinka su base64 enkodovani. Kao identifikator se najčešće koristi korisničko ime. Server će obraditi zahtjev samo ako može izvršiti validaciju identifikatora i lozinke za zaštićeni prostor specifikovan vrijednošću *realm* polja. Dodatni autentikacioni parametri se ne koriste. Na Slici 1 dat je dio koda Android aplikacije kojim se realizuje Basic autentikacija.

Na Slici 2 dati su dijelovi koda iOS aplikacije kojim se realizuje Basic autentikacija.

```
protected String doInBackground(String... params) {

    String URL = "http://192.168.0.100:8080/ServletSecureBasic/ServletBasic";
    String username = "admin";
    String password = "admin";
    String response = "";
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost request = new HttpPost(URL);

    try {
        HttpResponse responseFirst = httpClient.execute(request);
        if (responseFirst.getStatusLine().getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
            UsernamePasswordCredentials credentials = new UsernamePasswordCredentials(username, password);
            BasicScheme scheme = new BasicScheme();
            Header authorizationHeader = scheme.authenticate(credentials,request);
            request.addHeader(authorizationHeader);
            HttpEntity httpEntity = httpClient.execute(request).getEntity();
            response = EntityUtils.toString(httpEntity, "UTF-8");
        }else{
            HttpEntity httpEntity=responseFirst.getEntity();
            response = EntityUtils.toString(httpEntity, "UTF-8");
        }
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (AuthenticationException e) {
        e.printStackTrace();
    } finally {
        httpClient.getConnectionManager().shutdown();
    }
    return response;
}
```

Slika 1: *Dio koda Android aplikacije kojim se realizuje Basic autentikacija*

```
NSURL *url=[NSURL URLWithString:@"http://192.168.0.100:8080/ServletSecureBasic/ServletBasic"];
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData
                timeoutInterval:10.0];

[request setHTTPMethod:@"POST"];

[request setValue:base64Login forHTTPHeaderField:@"Authorization"];

NSURLConnection *urlConnection = [[NSURLConnection alloc] initWithRequest:request delegate:self];

//=========================================================================================================================================

- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
    if ([challenge previousFailureCount] == 0) {
        NSURLCredential *newCredential = [NSURLCredential credentialWithUser:[self preferencesName]
                                                                    password:[self preferencesPassword]
                                                                 persistence:NSURLCredentialPersistenceNone];
        [[challenge sender] useCredential:newCredential
                forAuthenticationChallenge:challenge];
    }
    else {
        [[challenge sender] cancelAuthenticationChallenge:challenge];
        [self showPreferencesCredentialsAreIncorrectPanel:self];
    }
```

Slika 2: *Dijelovi koda iOS aplikacije kojima se realizuje Basic autentikacija*

```
Frame 52: 1221 bytes on wire (9768 bits), 1221 bytes captured (9768 bits) on interface 1
Ethernet II, Src: IntelCor_af:17:2a (00:1f:3c:af:17:2a), Dst: SamsungE_75:91:e2 (88:32:9b:75:91:e2)
Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 47204 (47204), Seq: 1, Ack: 174, Len: 1155
Hypertext Transfer Protocol
  HTTP/1.1 401 Unauthorized\r\n
    Server: Apache-Coyote/1.1\r\n
    WWW-Authenticate: BASIC realm="protected"\r\n
    Content-Type: text/html;charset=ISO-8859-1\r\n
  Content-Length: 954\r\n
    Date: Wed, 24 Jul 2013 09:55:33 GMT\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.001307000 seconds]
    [Request in frame: 51]
Line-based text data: text/html
```

Slika 3: *Unauthorized odgovor od servera, Basic autentikacija*

```
Frame 59: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits) on interface 1
Ethernet II, Src: SamsungE_75:91:e2 (88:32:9b:75:91:e2), Dst: IntelCor_af:17:2a (00:1f:3c:af:17:2a)
Internet Protocol Version 4, Src: 192.168.0.103 (192.168.0.103), Dst: 192.168.0.100 (192.168.0.100)
Transmission Control Protocol, Src Port: 45521 (45521), Dst Port: http-alt (8080), Seq: 1, Ack: 1, Len: 212
Hypertext Transfer Protocol
  POST /ServletSecureBasic/ServletBasic HTTP/1.1\r\n
  Authorization: Basic YWRtaW46YWRtaW4=\r\n
      Credentials: admin:admin
  Content-Length: 0\r\n
    Host: 192.168.0.100:8080\r\n
    Connection: Keep-Alive\r\n
    User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)\r\n
    \r\n
    [Full request URI: http://192.168.0.100:8080/ServletSecureBasic/ServletBasic]
    [HTTP request 1/1]
    [Response in frame: 60]
```

Slika 4: *Zahtjev od klijenta sa Authorization zaglavljem, Basic autentikacija*

Na Slici 3 prikazan je Unauthorized odgovor od servera, a na Slici 4 novi zahtjev od klijenta sa Authorization zaglavljem.

Za bilježenje saobraćaja korišćen je Wireshark program, verzija 1.10.0. Ovaj program, kao i drugi programi iste ili slične namjene automatski dekoduju base64 enkodovane kredencijale (Wireshark dekodovane kredencijale prikazuje u okviru polja Credentials).

4.1.1.1 PREDNOSTI HTTP BASIC AUTENTIKACIJE

Prednost HTTP Basic autentikacije je da je dio HTTP specifikacije. Široko je podržana i jednostavna za implementaciju.

4.1.1.2. NEDOSTACI HTTP BASIC AUTENTIKACIJE

Najveći nedostatak HTTP Basic autentikacije je što se korisnički podaci prenose u otvorenom obliku preko mreže. base64 enkodovanje nije enkriptovanje, već samo vrsta enkodovanja koja binarne podatke konvertuje u ASCII format. Dekodovanje kredencijala je veoma jednostavna operacija. HTTP Basic autentikacija je podložna napadu tipa lažnog predstavljanja. Kod uspješno izvršenog napada ove vrste korisnik misli da komunicira sa legitimnim serverom a, zapravo, komunicira sa lažnim serverom ili *proxy*-jem koji prikuplja podatke od korisnika. HTTP Basic autentikacija se može koristiti u slučaju sakupljanja statističkih podataka i to u slučaju kada server obezbjeđuje korisničko ime i lozinku koji se ne mogu mijenjati. Basic autentikacija je podložna MITM napadu.

4.1.2. HTTP DIGEST AUTENTIKACIJA

Digest autentikacija [10] je, kao i Basic autentikacija, zasnovana na jednostavnom izazov-odgovor mehanizmu. Nakon što klijent zahtijeva pristup zaštićenom resursu, server šalje zahtjev za autentikaciju koji sadrži slučajnu vrijednost. Za generisanje slučajne vrijednosti može se iskoristiti npr. timestamp vrijednost, HTTP ETag zaglavlje i privatni ključ servera, od kojih bi bio kreiran digitalni sažetak. Server može u opcionom zaglavlju da specifikuje koji algoritam treba koristiti za izračunavanje digitalnog sažetka, a ako se to ne navede podrazumijevano se koristi MD5. Validan odgovor klijenta sadrži Authorization zaglavlje, za čiju vrijednost specifikuje naziv autentikacione šeme praćene sa korisničkim imenom, *realm* vrijednošću, primljenom slučajnom vrijednošću, URI-jem (*Uniform Resource Identifier*) zahtjeva i digitalnim sažetkom za čije izračunavanje se koriste korisničko ime,

```
protected String doInBackground(String... params) {

    String URL = "http://192.168.0.100:8080/ServletSecureDigest/ServletDigest";
    String username = "admin";
    String password = "admin";
    String response = "";
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost request = new HttpPost(URL);

    try {
        HttpResponse responseFirst = httpClient.execute(request);
        if (responseFirst.getStatusLine().getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
            Header wwwAuthenticateHeader = responseFirst.getFirstHeader("WWW-Authenticate");
            DigestScheme digestScheme = new DigestScheme();
            digestScheme.processChallenge(wwwAuthenticateHeader);
            UsernamePasswordCredentials credentials = new UsernamePasswordCredentials(username, password);
            request.addHeader(digestScheme.authenticate(credentials, request));
            HttpEntity httpEntity = httpClient.execute(request).getEntity();
            response = EntityUtils.toString(httpEntity, "UTF-8");
        }else{
            HttpEntity httpEntity=responseFirst.getEntity();
            response = EntityUtils.toString(httpEntity, "UTF-8");
        }
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (MalformedChallengeException e) {
        e.printStackTrace();
    } catch (AuthenticationException e) {
        e.printStackTrace();
    } finally {
        httpClient.getConnectionManager().shutdown();
    }
    return response;
}
```

Slika 5: *Dio koda iz Android aplikacije kojim se realizuje Digest autentikacija*

```
Frame 29: 1318 bytes on wire (10544 bits), 1318 bytes captured (10544 bits) on interface 1
Ethernet II, Src: IntelCor_af:17:2a (00:1f:3c:af:17:2a), Dst: SamsungE_75:91:e2 (88:32:9b:75:91:e2)
Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.103 (192.168.0.103)
Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 44806 (44806), Seq: 1, Ack: 176, Len: 1252
Hypertext Transfer Protocol
  HTTP/1.1 401 Unauthorized\r\n
  Server: Apache-Coyote/1.1\r\n
  WWW-Authenticate: Digest realm="primjer.com",qop="auth",nonce="eb7c18b9ff26a4358c9783b8a5b99ee0",opaque="f877a195df4fd6ce9fbb217268893cee"\r\n
  Content-Type: text/html;charset=ISO-8859-1\r\n
  Content-Length: 954\r\n
  Date: Wed, 24 Jul 2013 10:00:46 GMT\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.001679000 seconds]
  [Request in frame: 28]
Line-based text data: text/html
```

Slika 6: *Unauthorized odgovor od servera, Digest autentikacija*

Slika 7: *Zahtjev od klijenta sa Authorization zaglavljem, Digest autentikacija*

realm, lozinka, HTTP metoda i URI zahtjeva. Kod ovog tipa autentikacije lozinka se nikada ne šalje u otvorenom tekstu. Na Slici 5 dat je dio koda iz Android aplikacije kojim se realizuje Digest autentikacija.

Na Slici 6 prikazan je Unauthorized odgovor servera.

Na Slici 7 dat je novi zahtjev od klijenta sa Authorization zaglavljem, za slučaj HTTP Digest autentikacije.

4.1.2.1. PREDNOSTI HTTP DIGEST AUTENTIKACIJE

Digest Autentikacija je dizajnirana da otkloni većinu nedostataka Basic autentikacije. Ipak, ova šema ne obezbjeđuje enkripciju sadržaja koji se prenosi, već samo da se lozinka ne prenosi u otvorenom tekstu. Svi ostali podaci su dostupni potencijalnim napadačima. Server može onemogućiti ponovno korišćenje iste slučajne vrijednosti ili digest vrijednosti da bi spriječio napad ponavljanjem istih poruka. Takođe, ako se u slučajnu vrijednost uključi timestamp može se odrediti i vremenski period validnosti slučajne vrijednosti. Kada se koristi Digest autentikacija moguće je izvesti napad lažnim predstavljanjem, ali ga je teže sprovesti nego kod Basic autentikacije.

4.1.2.2. NEDOSTACI HTTP DIGEST AUTENTIKACIJE

Po savremenim kriptografskim standardima Digest autentikacija nudi slabu zaštitu. Zaštita koju pruža ova autentikacija zavisi od načina implementacije, jer je veliki broj opcija proizvoljan. Npr. slučajnu vrijednost je moguće konstruisati tako da se onemoguće napadi ponavljanjem poruka, ali to zahtjeva da server pamti sve slučajne vrijednosti, što će rezultirati nešto lošijim performansama servera. Digest autentikacija je ranjiva na *man in the middle* napad, kao i Basic autentikacija. Takođe, moguće je da napadač prisluškuje komunikaciju i na osnovu riječi iz rječnika i podataka koji se prenose (slučajna vrijednost, korisničko ime, i sl.) pokuša pogoditi lozinku.. Ako je moguće da maliciozni server bira slučajnu vrijednost koju će klijent koristiti za računanje odgovora, onda je riječ o "*chosen plaintext*" napadu. Uz pomoć ovog napada moguće je sakupiti sve odgovore koji koriste istu slučajnu vrijednost što olakšava *brute force* napad. Uz pomoć "*chosen plaintext*" napada moguće je unaprijed izračunati odgovor za određeni skup lozinki, pa samo upoređivati odgovor sa izračunatom vrijednošću. Ovo je tzv. *precomuted dictonary napad*. Da bi se spriječili *chosen plaintext, brute force i precomuted dictonary* napadi, klijent treba koristiti opcioni *cnonce* atribut koji predstavlja slučajnu vrijednost koju klijent postavlja, što onemogućava napadača da unaprijed izračuna moguće odgovore.

4.2. *HTTPS PROTOKOL*

Da bi se obezbijedila sigurna komunikacija između mobilne aplikacije i servera može da se koristi HTTPS (HTTP *Secure*) protokol. HTTPS predstavlja kombinaciju HTTP i SSL (*Secure Sockets Layer*) ili TLS (*Transport Layer Security*) protokola [11]. TLS je verzija SSL-a koju je standardizovao IETF (*Internet Engineering Task Force*). Ne postoji fundamentalna razlika u korišćenju HTTP protokola preko SSL-a ili TLS-a. SSL omogućava autentikaciju servera i klijenta i zaštićenu komunikaciju između njih. Nakon što je uspostavljena sigurna konekcija između klijenta i servera, podaci koji se razmjenjuju su enkriptovani primjenom simetričnih algoritama. SSL obezbjeđuje mehanizam za sigurno zatvaranje konekcije. Kada se primi obavještenje da se konekcija želi zatvoriti, ta strana u komunikaciji više neće primati podatake. SSL implementacija mora inicirati slanje obavještenja o zatvaranju konekcije prije samog zatvaranja. Moguće je da strana u komunikaciji nakon slanja ovog obavještenja, prije nego što druga strana odgovori, zatvori konekciju što predstavlja "nepotpuno zatvaranje". U ovom slučaju moguće je ponovo upotrebiti sesiju. Ako se konekcija zatvori prije slanja obavještenja sigurnost podataka nije ugrožena, ali je moguće da se dio podataka izgubi, pa je potrebno provjeriti da li su svi podaci primljeni. Autentikacija servera je obavezna da bi klijent bio siguran sa kim komunicira. Autentikacija se obavlja u 4 koraka: provjerava se da li je sertifikat servera važeći, da li se vjeruje sertifikacionom tijelu (CA, eng. *Certification Authority*) koje je izdalo sertifikat, da li je digitalni potpis u sertifikatu validan i da li se ime domena servera sa kojim klijent komunicira poklapa sa imenom u sertifikatu. Posljednji korak obezbjeđuje zaštitu od *man in the middle* napada. Ako se autentikacija ne obavi uspješno, klijent treba da obavijesti korisnika da se ne može utvrditi identitet servera ili da prekine konekciju zbog greške sa sertifikatom. Klijentska autentikacija nije obavezna. Ako je ona potrebna, obavlja se na isti način kao autentikacija servera. Primjer uspostavljanja HTTPS konekcije od strane Android aplikacije prema serveru je dat na Slici 8 [12].

Ukoliko se ne uspije uspostaviti sigurna konekcija desiće se SSLHandshakeException. Razlozi za to mogu biti: Android ne vjeruje CA-u koje je izdalo sertifikat serveru, sertifikat je samopotpisan, lanac povjerenja ne postoji ili nije potpun. Razvojni programeri često odluče da njihova aplikacija vjeruje svim sertifikatima, što je pogrešna praksa, jer tada SSL ne pruža nikakvu zaštitu. Sigurnija varijanta je da se napravi aplikacija koja vjeruje određenom sertifikatu ili grupi sertifikata. Na Slici 9 je prikazano kako se može uspostaviti HTTPS konekcija u slučaju da je sertifikat servera samo-

```
protected String doInBackground(String... params) {

    String URL = "https://192.168.0.100:8443/ServletSecureSSL/ServletSSL";
    String response = "";

    try {
        URL url = new URL(URL);
        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        InputStream in = urlConnection.getInputStream();
        response = readStream(in);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return response;
}
```

Slika 8: *Uspostavljanje HTTPS konekcije iz Android aplikacije*

```
protected String doInBackground(String... params) {
    try {
        String URL = "https://192.168.0.100:8443/ServletSecureSSL/ServletSSL";
        String response = "";

        CertificateFactory cf = CertificateFactory.getInstance("X.509");
        InputStream caInput = new BufferedInputStream(new FileInputStream("sertifikat.cer"));
        try {
            Certificate ca = cf.generateCertificate(caInput);
        } finally {
            caInput.close();
        }
        String keyStoreType = KeyStore.getDefaultType();
        KeyStore keyStore = KeyStore.getInstance(keyStoreType);
        keyStore.load(null, null);
        keyStore.setCertificateEntry("ca", ca);

        String tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
        TrustManagerFactory tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
        tmf.init(keyStore);

        SSLContext context = SSLContext.getInstance("TLS");
        context.init(null, tmf.getTrustManagers(), null);

        URL url = new URL(URL);
        HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();
        urlConnection.setSSLSocketFactory(context.getSocketFactory());
        InputStream in = urlConnection.getInputStream();
        response = readStream(in);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (CertificateException e) {
        e.printStackTrace();
    } catch (KeyStoreException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (KeyManagementException e) {
        e.printStackTrace();
    }
    return response;
}
```

Slika 9: *Uspostavljanje HTTPS konekcije u slučaju samopotpisanog sertifikata servera*

potpisan [12]. Kao što se može primjetiti iz priloženog koda programski se kreira key store koji sadrži sertifikat "ca" koji je samopotpisan. Kreira se TrustManager koji se inicijalizuje na osnovu key store-a koji sadrži "ca" sertifikat . TrustManager kreiran na ovaj način vjeruje sertifikatu koji se nalazi u key store-u. Sistem koristi TrustManager u procesu validacije sertifikata dobijenog od strane servera. Na osnovu kreiranog TrustManager-a inicijalizuje se SSLContext na osnovu koga se može dobiti SSLSocketFactory koja će se koristiti pri uspostavljanju HTTPS konekcije. Ovaj pristup se, uz manje izmjene, može koristiti i u slučaju kada Android ne vjeruje CA-u koje je izdalo sertifikat serveru ili lanac povjerenja ne postoji ili nije potpun.

4.2.1. PREDNOSTI HTTPS PROTOKOLA

Kada se koristi HTTPS protokol tada su enkriptovani sljedeći elementi komunikacije [2]:

- URL zahtjeva
- sadržaj odgovora
- podaci koji se šalju od klijenta do servera i obratno
- kolačići koji se razmjenjuju
- sadržaj HTTP zaglavlja

SSL omogućava da klijent i server dogovore algoritme i ostale kriptografske parametre koje će koristiti u komunikaciji, što olakšava interoperabilnost i omogućava laku nadogradnju

protokola uvođenjem novih, snažnijih algoritama. Klijent i server će komunicirati koristeći najsnažniji algoritam koji obe strane podržavaju.

4.2.2. NEDOSTACI HTTPS PROTOKOLA

Sigurnost SSL-a zavisi od kriptografskog algoritma koji se koristi u određenoj sesiji, pri čemu SSL nasljeđuje eventualna ograničenja korišćenog algoritama. Serverska strana može podržavati upotrebu širokog spektra algoritama da bi što više klijenata moglo pristupati serveru. To je prednost, ali ujedno i opasnost, jer klijent može izabrati algoritam koji ne obezbjeđuje dovoljnu sigurnost [13]. SSL obezbjeđuje tajnost i integritet podataka u prenosu, autentikaciju servera i klijenta, ali ne obezbjeđuje neporecivost. Takođe, ne postoji podrška za UDP (*User Datagram Protocol*), samo za TCP (*Transmission Control Protocol*). Pošto i klijent i server vrše enkripciju i dekripciju performanse mogu biti loše. Pored toga, ne postoji mogućnost zaštite jednog dijela poruke, jer se enkriptuje čitav prenosni kanal. Bitno je reći i to da SSL obezbjeđuje *point-to-point* zaštitu, što znači da se podaci na krajnjim tačkama (klijent i server) nalaze u otvorenom obliku. Isto tako, SSL nije pogodan za komunikaciju između više korisnika, pri čemu postoje posredničke tačke, jer će se svi podaci koji se razmjenjuju u ovim tačkama naći u otvorenom obliku.

4.3. *ZAŠTITA NA NIVOU PORUKE*

Zaštita na nivou poruke se koristi za siguran prenos poruka između klijenta i servera. Fokusirana je na individualne poruke koje se prenose kroz mrežu. Za zaštitu poruka koriste se mehanizmi kao što su enkripcija i digitalno potpisivanje koji se primjenjuju direktno na sadržaj poruke. Sve sigurnosne informacije su enkapsulirane u poruku. Na taj način poruka će biti bezbjedna, čak i u slučaju da se prenosi preko nezaštićenog protokola kao što je HTTP. Zaštita na nivou poruke pripada aplikacionom sloju. Dok SSL nudi *point-to-point* sigurnu vezu, zaštita na nivou poruke obezbjeđuje *end-to-end* sigurnost, tj. zaštitu od klijenta do servera, bez obzira na broj posredničkih tačaka koje prihvataju istu poruku.

U slučaju kada je dovoljno obezbijediti *point-to-point* sigurnu vezu, može se koristiti SSL. Problem nastaje kada između klijenta i servera postoje posrednici koji mogu ugroziti sigurnost podataka, kao što je npr. *proxy* servis/server. U takvim slučajevima između klijenta i servera potrebno je praviti nove SSL konekcije. *Proxy* server mora uspostaviti SSL konekcije i prema klijentu i prema serveru. On na osnovu poruke zna kome je upućena i prosljeđuje je preko nove SSL konekcije. Pošto posrednik može pristupiti sadržaju poruke sigurnost podataka može biti ugrožena. U slučaju da se koristi zaštita na nivou poruke, kada poruka prolazi preko posrednika enkriptovani sadržaj ostaje nepoznat za njih, sve dok poruka ne stigne do servera kome je namijenjena.

Jedna od specifikacija koja se odnosi na zaštitu na nivou poruke, u oblasti SOAP (*Simple Object Access Protocol*) baziranih web servisa, jeste WS-Security specifikacija [14]. Trenutno, Android ne podržava SOAP bazirane web servise, pa samim tim ni WS-Security specifikacija nije podržana u aktuelnim verzijama Android API-ja. Za potrebe ovog rada, korištena je verzija kSOAP2 biblioteke koja je portovana na Android platformu. Bitno je napomenuti da kSOAP2 ne podržava sigurnosne osobine SOAP baziranih web servisa, kako za zaštitu na transportnom, tako i za zaštitu na aplikativnom sloju. Tako je, da bi se obezbijedila zaštita na nivou poruke, neophodno izvršiti proširenje kSOAP2 biblioteke. Primjer jednog takvog proširenja dat je u [15].

4.3.1. PREDNOSTI ZAŠTITE NA NIVOU PORUKE

Kada se koristi zaštita na nivou poruke, tada je poruke moguće slati preko bilo kog transportnog protokola, što nije slučaj sa HTTPS-om. Prednost ovog tipa zaštite nad HTTPS-om je i veća fleksibilnost. Ako tajnost podataka nije bitna već samo npr. integritet, autentičnost ili neporecivost, poruku je moguće samo digitalno potpisati. Takođe, moguće je zaštiti samo dio poruke. Kao što je ranije i rečeno, obezbjeđena je *end-to-end* sigurnost, pa posrednici ne mogu da utiču na sigurnost poruke.

4.3.2. NEDOSTACI ZAŠTITE NA NIVOU PORUKE

Iako je sadržaj zaštićen enkripcijom, podaci o tome ko šalje poruku i kome je šalje su vidljivi. Zaštita na nivou poruka unosi značajan *overhead*, pa su performanse lošije. Potrebno je da klijent i server podržavaju standarde za zaštitu na nivou poruka, što može smanjiti interoperabilnost.

5. ZAKLJUČAK

Pošto napadi na podatke koje razmijenjuju mobilne aplikacije i udaljeni serveri mogu nanijeti veliku štetu vlasnicima podataka, neophodno je te podatke zaštiti na adekvatan način. Izbor mehanizma zaštite zavisi od mnogo faktora, poput zahtijevanog nivoa sigurnosti, zahtijevanih performansi sistema i/ili kompleksnosti/jednostavnosti implementacije. Zbog toga je potrebno odrediti prioritete i shodno tome odabrati odgovarajući mehanizam zaštite.

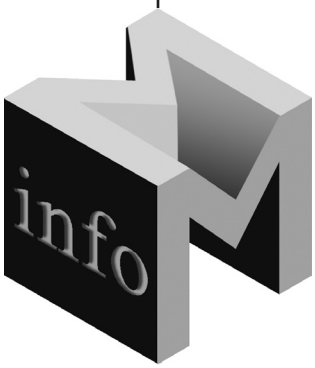
Kao što je ranije rečeno HTTP Basic i HTTP Digest autentikacija omogućavaju samo da se obezbijedi kontrola pristupa resursima na serveru. HTTPS omogućava uspostavljanje sigurne konekcije, kao i autentikaciju klijenta i servera pomoću digitalnih sertifikata. Zaštita na nivou poruke kao što joj samo ime kaže, štiti poruku, i obezbjeđuje da poruka sigurno stigne od servera ka klijentu, i obrnuto. Mehanizmi zaštite obrađivani u ovom radu nalaze se na aplikativnom i transportnom sloju OSI referentnog modela. Postoje mehanizmi zaštite i na drugim slojevima, od kojih je najpoznatiji IPSec na mrežnom sloju koji omogućava uspostavljanje sigurnog tunela za komunikaciju. Iako je riječ o veoma značajnim mehanizmima zaštite, oni se znatno manje koriste od mehanizama opisanih u ovom radu, zbog nepostojanja adekvatne podrške u API-jima za razvoj mobilnih aplikacija.

6. LITERATURA

[1] *R. Shirey, „Internet Security Glossary“, RFC 2828, May 2010.*
[2] *W. Stallings, Cryptography and Network Security Principles and Practice, Fifth Edition, January 2010.*
[3] *A. Biryukov, J. Großschädl, „Cryptanalysis of the Full AES Using GPU-Like Special-Purpose Hardware“, Fundamenta Informaticae, 114 3-4, 221-237, 2012*
[4] *H. C. A. van Tilborg, S. Jajodia (Eds.): Encyclopedia of Cryptography and Security, 2nd Ed. Springer 2011.*
[5] *A. J. Menezes , S. A. Vanstone , P. C. Van Oorschot, Handbook of Applied Cryptography, CRC Press, Inc., Boca Raton, FL, 1996*
[6] *W. Stallings, Network and Internetwork Security: Principles and Practice, Prentice Hall, Chapter 1, 1995.*
[7] *E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, „Recommendation for Key Management“, Special Publication 800-57 Part 1 Rev. 3, NIST, July 2012.*
[8] *Connecting to the Network, http://developer.android.com/training/basics/network-ops/connecting.html, posjećeno 27.07.2013.*
[9] *HTTP 1.0, http://www.w3.org/Protocols/HTTP/1.0/spec.html, posjećeno 14.07.2013.*
[10] *J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, „HTTP Authentication: Basic and Digest Access Authentication“, RFC 2617, June 1999.*
[11] *E. Rescorla, „HTTP over TLS“, RFC 2818, May 2000.*
[12] *Security with HTTPS and SSL, http://developer.android.com/training/articles/security-ssl.html, posjećeno 15.07.2013.*
[13] *H. K. Lee, T. Malkin, E. Nahum, „Cryptographic Strength of SSL/TLS Servers:Current and Recent Practices“, October 2007.*
[14] *B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon, „Web Services Security (WS-Security)“, April 2002.*
[15] *C. Kleiner, T. Schneider, Securing soap web services for mobile devices on different platforms, in: Proceedings of the 6th conference MMS, Vol. 185 of LNI, Springer, 2011, pp. 25-38*

Danijela Zoran
Elektrotehnički fakultet Banjaluka, RS, BiH
Kontakt: danijelazoran000@gmail.com
Oblasti interesovanja: Android platforma, sigurnost, objektno-orijentisano programiranje i projektovanje

doc. dr Zoran Đurić
Elektrotehnički fakultet Banjaluka, RS, BiH
Kontakt: zoran.djuric@etfbl.net
Oblasti interesovanja: sigurnost, kriptografija, PKI, platni sistemi i protokoli, formalna verifikacija, objektno-orijentisano programiranje i modelovanje, internet programiranje, XML-bazirana međuoperativnost, Web servisi, penetration testing



infoM

UPUTSTVO ZA PRIPREMU RADA

1. Tekst pripremiti kao Word dokument, A4, u kodnom rasporedu 1250 latinica ili 1251 ćirilica, na srpskom jeziku, bez slika. Preporučen obim – oko 10 strana, single prored, font 11.
2. Naslov, abstakt (100-250 reči) i ključne reči (3-10) dati na srpskom i engleskom jeziku.
3. Jedino formatiranje teksta je normal, bold, italic i bolditalic, VELIKA i mala slova (tekst se naknadno prelama).
4. Mesta gde treba ubaciti slike, naglasiti u tekstu (Slika1...)
5. Slike pripremiti odvojeno, VAN teksta, imenovati ih kao u tekstu, radi identifikacije, u sledećim formatima: rasterske slike: jpg, tif, psd, u rezoluciji 300 dpi 1:1 (fotografije, ekranski prikazi i sl.), vektorske slike – cdr, ai, fh,eps (šeme i grafikoni).
6. Autor(i) treba da obavezno priloži svoju fotografiju (jpg oko 50 Kb), navede instituciju u kojoj radi, kontakt i 2-4 oblasti kojima se bavi.
7. Maksimalni broj autora po jednom radu je 5.

Redakcija časopisa Info M