Secure data processing application

Kelly Lunghamer and Siddharth Sharma

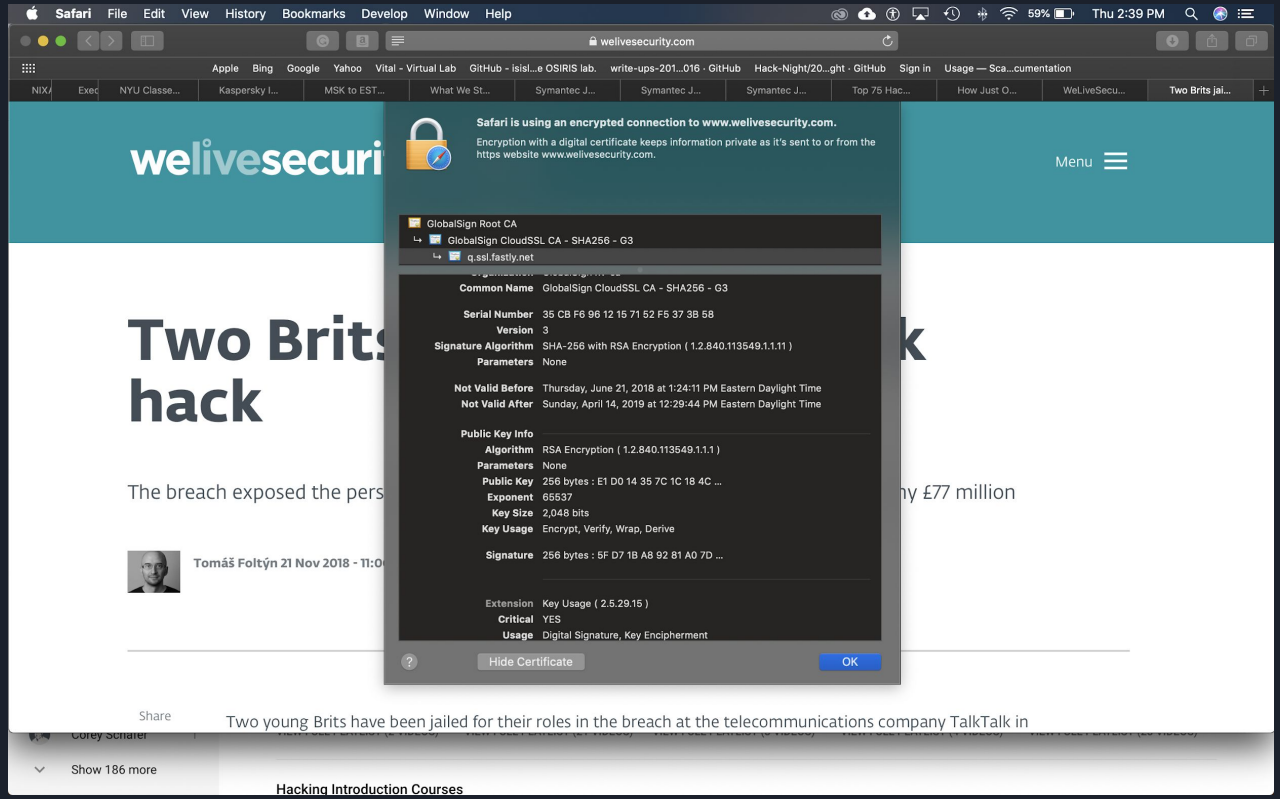
Introduction

- Application is a blog post, with an extra layer security achieved using Hmac and nonce.
  - Hmac protects the data integrity
  - Nonces protects against the replay attacks
- Passwords are protected using salts
- We used industry standard encryption methods, such as AES with PBKDF2 to encrypt the post.
- User can sign up, login, and create blog posts, while no two users can post the same content to protect intellectual property

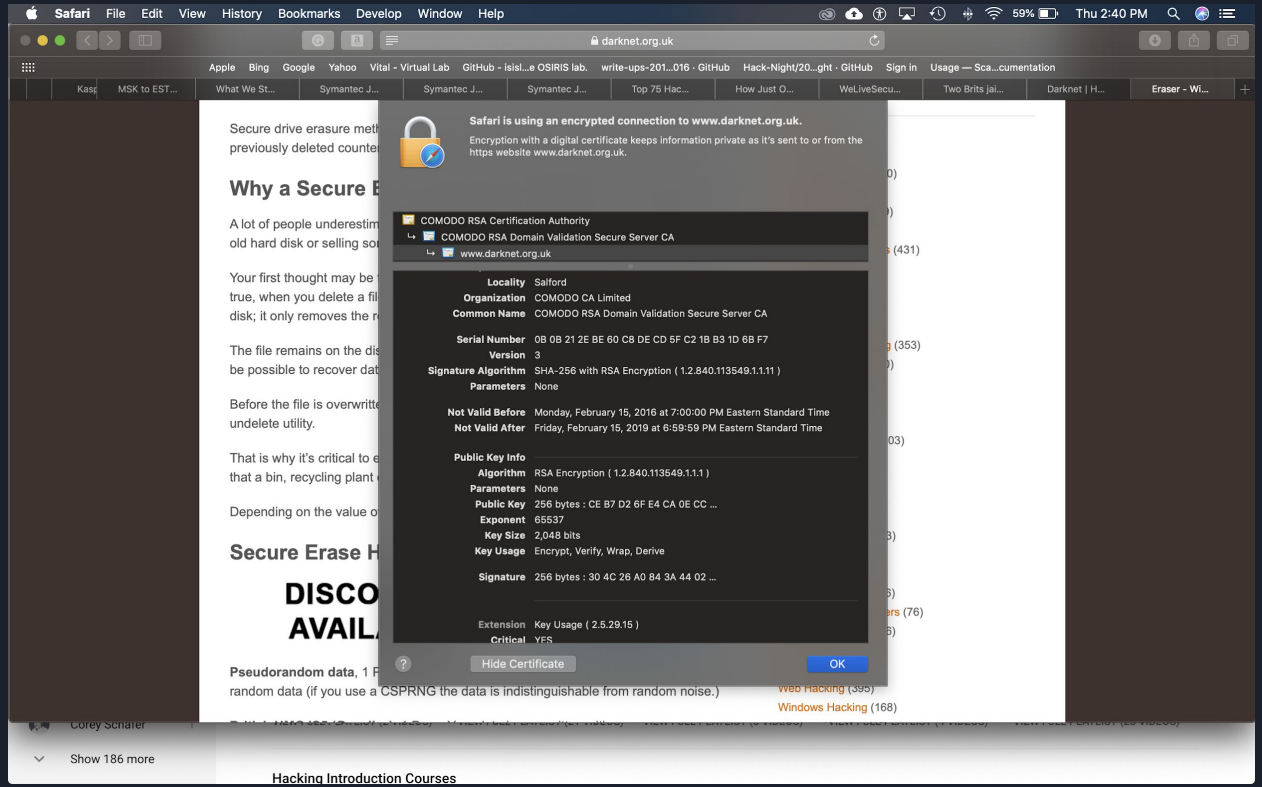
Phases of security implementation

- Phase 1: Setting up a secure connection
  - RSA, AES 256, SHA256
- Phase 2: Login
  - One-way functions, Oracle
- Phase 3: Post a blog
  - AES with PBKDF2, HMAC

Phase 1: PKI examples



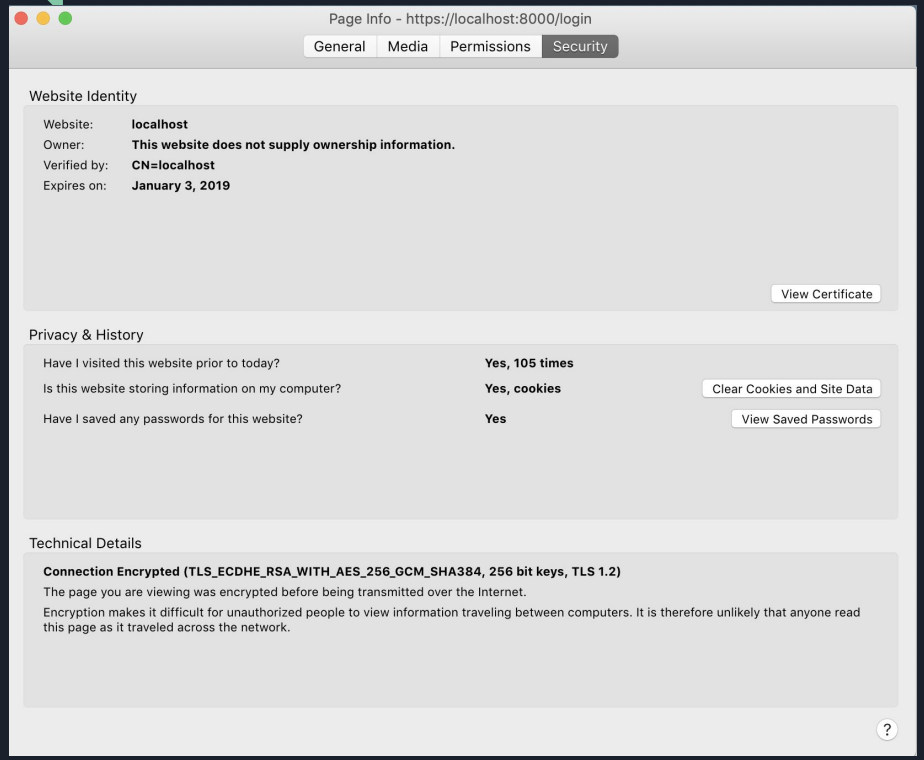
Phase 1: PKI examples (cont.)



Phase 1: Certification generation

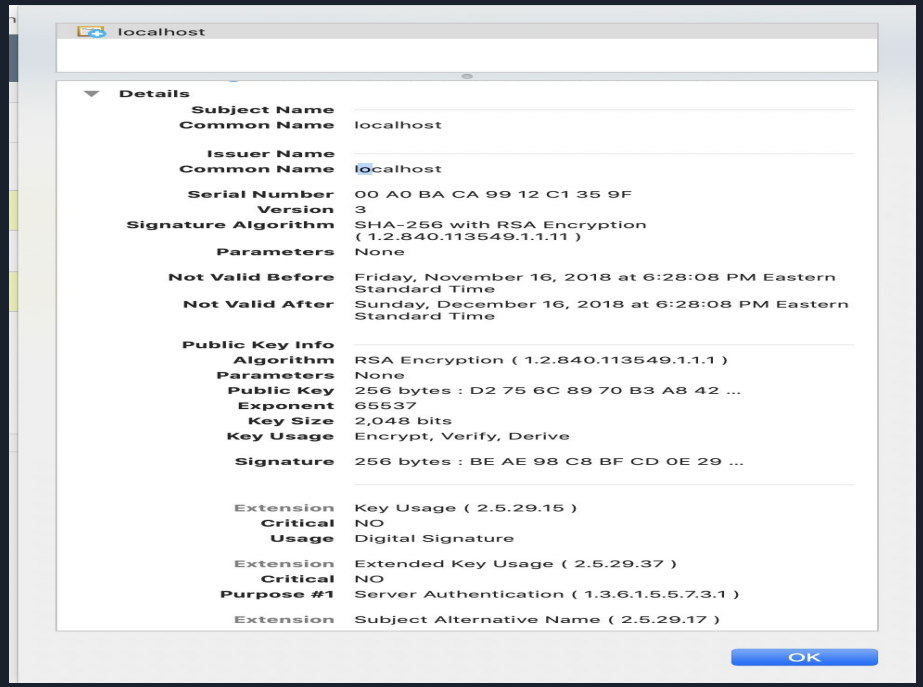
openssl req -x509 -out localhost.crt -keyout localhost.key -newkey rs**a:2048** -nodes **-sha256** -subj '/CN=localhost' -extensions EXT -config <(\ printf "[dn]\nCN=localhost\n[req]\ndistinguished_name = dn\n[EXT]\nsubjectAltName=DNS:localhost\nkeyUsage=digitalSignature\nextendedKeyUsage =serverAuth")

Phase 1: PKI Certificate

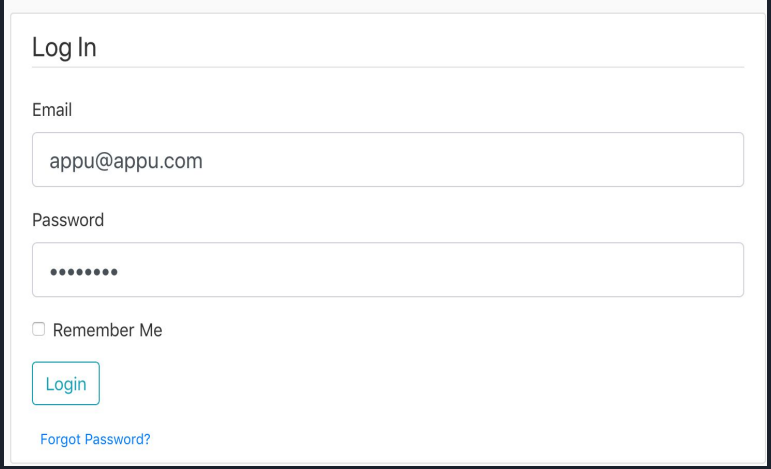


- Cipher suite:
  - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
  - ECDHE guarantees perfect forward secrecy
  - key exchange - RSA 2048
  - encryption - AES 256 **with GCM**
  - hashing - SHA256

Phase 1: PKI Certificate (cont.)



Phase 2: Login



- bcrypt is a one-way function that creates a hash of the password
- It also uses a salt that is randomly generated
- Given a hash and the user-inputted password, an oracle can verify a successful login at the server

Phase 2: Code using bcrypt

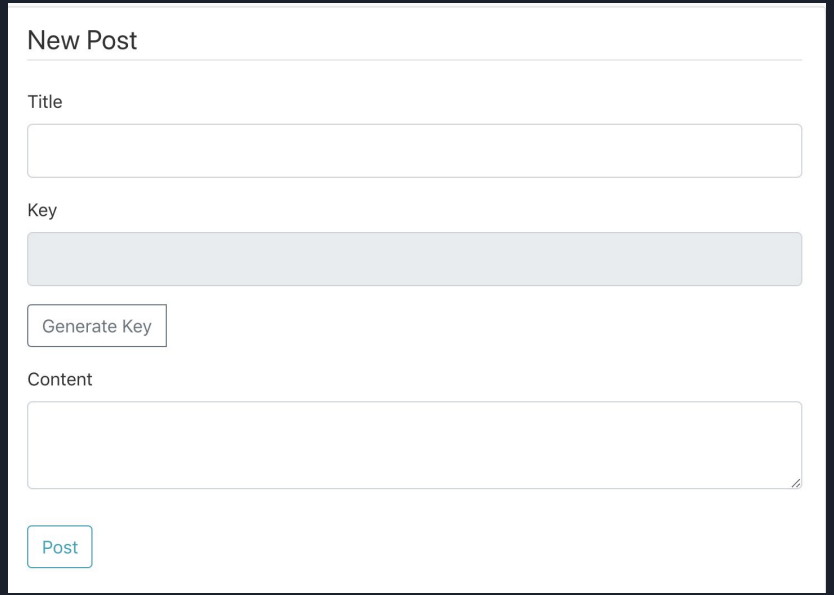
```
form = RegistrationForm()
if form.validate_on_submit():
    hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
    user = User(username=form.username.data, email=form.email.data, password=hashed_password)
    db.session.add(user)
    db.session.commit()
    flash('Your account has been created! You are now able to log in', 'success')
    return redirect(url_for('login'))
return render_template('register.html', title='Register', form=form)
```

Phase 2: Code using bcrypt (cont.)

```
user = User.query.filter_by(email=form.email.data).first()
if user and bcrypt.check_password_hash(user.password, form.password.data):
    login_user(user, remember=form.remember.data)
    next_page = request.args.get('next')
    return redirect(next_page) if next_page else redirect(url_for('home'))
```

Phase 3: Blog post

- All blog posts' title and content are encrypted on the client side.
- AES keys never travel over the internet, hence are not prone to any kind an attack.
- We are appending our own SHA256 Hmac with our data, to add an extra layer of security to protect against unauthorized data manipulation.
- We are using a nonce to protect against replay attacks.



Phase 3: Protecting IP & Nonces

```
replay_token = generate_iv()
hmac = Replay(text=replay_token, used=0)
db.session.add(hmac)
db.session.commit()
return render_template('create_post.html', title='New Post',
                       form=form, legend='New Post', replay_token=replay_token)
```

```
title = hash_sha256(data['title'])

res = Hmac.query.filter_by(title=title).all()
if res:
    return '1'

content = hash_sha256(data['content'])
res = Hmac.qu                          content).all()
if res:
    return '2
```

Definition:
flaskblog/routes.py:295

Security considerations

- Confidentiality
  - The connection is encrypted using AES, protecting confidentiality
  - ECDHE also provides perfect forward secrecy and prevents the compromise of a long-term secret key from affecting the confidentiality of past conversations
  - Our login procedure provides authenticity and confidentiality
  - Blog post keys on the client side, so they cannot be stolen over the network
- Integrity attack detection
  - SHA256 is used for HMAC, which protect data integrity and authenticity
  - All blog posts to the server append an HMAC and which is validate by the server
  - Users who are not logged on or registered cannot post to the blog site
- Efficiency
  - Used 2048 instead of 4096 to avoid extra CPU consumption without compromise to security
  - bcrypt is efficient, because it uses Blowfish - which provides a quick encryption rate
  - All the encryption/decryption of the blog posts are conducted on the client side increase efficiency

Potential attacks/adversaries

- If an adversary is eavesdropping, they can get the encrypted data but could only brute force; we are using AES 256 with GCM (Galois counter mode)
  - GCM is widely adopted because of its efficiency and performance; it also provides data authenticity and confidentiality
  - AES proves protection against eavesdropping
- Chosen message attacks and chosen ciphertext attacks
  - AES with GCM will protect us attack these attacks.
- Protection against replay attacks are provided using nonces
- There is a potential for brute force, but it’s not feasible considering time
  - We also used PBKDF2 for blog posts, which prevents brute forcing

References

1. https://www.ibm.com/support/knowledgecenter/bg/SSKTMJ_9.0.1/admin/conf_port_enc_adv_r.html
2. https://en.wikipedia.org/wiki/PBKDF2