ReST Code Patterns

Dallo sviluppo al rilascio

LUCA MINUTI
DEVELOPER

email
LUCA.MINUTI@GMAIL.COM
GITHUB
HTTPS://GITHUB.COM/LMINUTI/

DelphiDay
italian conference

wintech
italia

GITHUB PROJECTS

WIRL
https://github.com/Delphi-Blocks/WiRL

GRAPHQL FOR DELPHI
https://github.com/lminuti/graphql

DELPHI-OPENSSL
https://github.com/lminuti/delphi-openssl

DELPHI-WKHTMLTOX
https://github.com/lminuti/delphi-wkhtmltox

DelphiDay
italian conference

wintech
italia

AGENDA

- Sicurezza
  - Autenticazione e Autorizzazione (OAuth2, SAML, SPID, JWT)
  - Injection e cross-site scripting (XSS)
  - Gestione delle password
- Robustezza e scalabilità
  - Thread safety (DB, pooling)
  - Stateless vs stateful
  - Load Balancing e configurazione del WebServer
- Debug e produzione
  - Servizi, Logging, File Server

DelphiDay italian conference

Sicurezza

1

UN PO' DI DATI

https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022

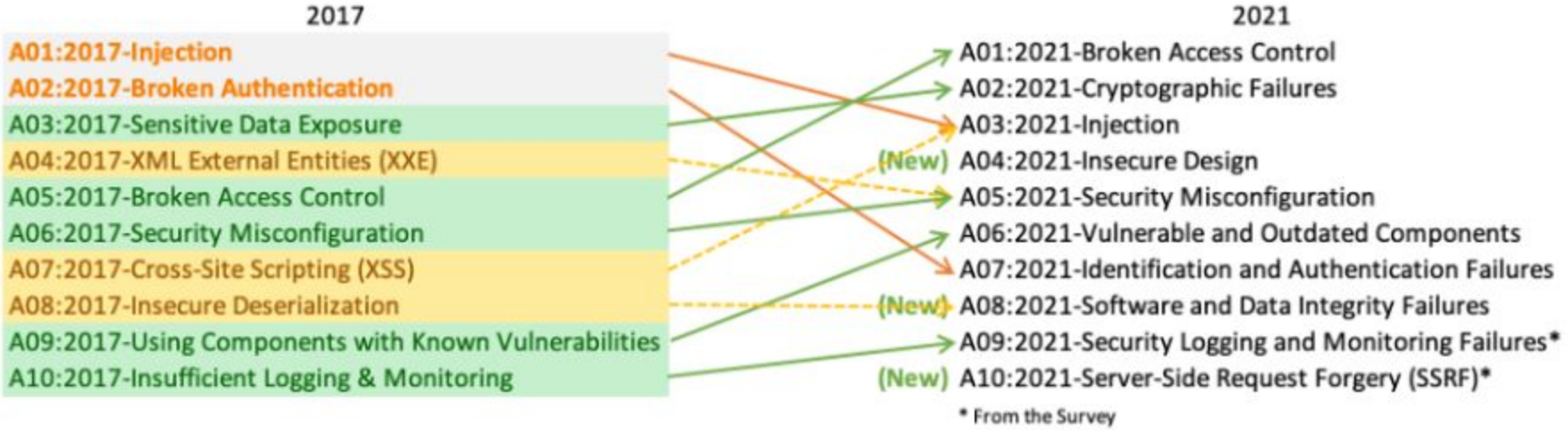
THE 9 TOP THREATS

- Ransomware;
- Malware;
- Social engineering;
- Threats against data;
- Threats against availability and integrity;
- Disinformation – misinformation;
- Supply-chain attacks

THREAT ACTOR TRENDS

- State-sponsored actors
- Cybercrime actors
- Hacker-for-hire actors
- Hacktivists

UN PO' DI DATI


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

OWASP

- L'Open Web Application Security Project
- Progetto open-source per la sicurezza delle applicazioni
- Nato il 9 settembre 2001
- Guide e consigli sulla creazione di applicazioni Internet sicure
- Suite di test (proxy tool: zap)
- Top 10 vulnerability

https://owasp.org/

BROKEN ACCESS CONTROL

- Con “access control” si intende quel processo che fa un modo che l'utente possa visualizzare solo le risorse a lui destinate
- Le vulnerabilità più comuni sono rese possibili da:
  - Mancato controllo degli URL
  - Mancato controllo sulle risorse server side
  - Visualizzazione di dati di terzi tramite id univoco (id utente)
  - Modifica del token JWT o del cookie di sessione

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

demo time

CRYPTOGRAPHIC FAILURES

- Questa vulnerabilità riguarda il trattamento dei dati particolarmente sensibili (password, numeri di carta di credito, dati personali ecc.). Per tutti questi dati:
  - Sono trasmessi in chiaro?
  - Sono cifrati con algoritmi datati o peggio “fai da te”?
  - Eventuali chiavi sono salvate nel VCS?
  - Sono stati usati MD5 o SHA1 come hash in ambito crittografico?

https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

INJECTION

- Un’applicazione è vulnerabile se:
  - I dati forniti dall’utente non sono validati o “sanitized”
  - L’input dell’utente viene usato per costruire query dinamiche o passate a qualche forma di interprete (eval)
  - L’input dell’utente è usato per creare dei filtri con ORM o simili

https://owasp.org/Top10/A03_2021-Injection/

SQL INJECTION

- Tecnica usata per attaccare applicazioni di gestione dati
- Sia applicazioni Web che desktop
- Innietta codice SQL malevolo
- Query costruita dal programma:

```
"SELECT * FROM users WHERE name = '" + userName + "'"
```

- Parametri:

```
userName = “' OR '1'='1 “
```

https://owasp.org/Top10/A03_2021-Injection/

NON SOLO SQL

- HTML injection
- Code injection (eval, scripting engine)
- XPath
- Nomi di file
- Accesso alle risorse
- Nomi di processi

https://owasp.org/Top10/A03_2021-Injection/

demo time

Autenticazione 2

DEFINIZIONI

- Autenticazione: chi sei tu?
  - User/password
  - Certificati, firma digitale
  - SSO: SAML, OpenID, LDAP, OAuth2
  - HTTP Auth: Basic, digest
- Autorizzazione: cosa puoi fare?
  - Ruoli
  - Privilegi
  - JWT

AUTENTICAZIONE

- Server to server authentication
  - Fiducia all'interno del perimetro (pericoloso per MiTM), usare almeno HTTPS
  - Basic Auth (HTTPS!), Digest Auth
  - Certificati client (più certificati, più complessità)
  - OAuth2
  - OpenID-Connect

Basic authentication

```
# Traffico HTTP tra client e server


GET /private/ HTTP/1.1
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="ACME"

# Il client calcola il token di sicurezza come base64 di
“utente:password”

GET /private/ HTTP/1.1
Authorization: Basic aHR0cHdhdGNoOmY=
HTTP/1.1 200 Ok
```

Digest authentication (simple)

```
# Traffico HTTP tra client e server


GET /private/ HTTP/1.1
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="ACME", nonce="1234"

# Il client calcola il token di sicurezza:
# HA1 = MD5(username:realm:password)
# HA2 = MD5(method:digestURI)
# response = MD5(HA1:nonce:HA2)

GET /private/ HTTP/1.1
Authorization: Digest username="Luca", realm="ACME",
response="...", nonce="1234"
HTTP/1.1 200 Ok
```

VANTAGGI DIGEST

- La password non viene usata direttamente
- Client nonce previene chosen-plaintext attacks come rainbow tables
- Server nonce può contenere timestamps (previene replay attacks)
- Il server può mantenere una lista di server nonce usare per prevenirne il riuso

LA PASSWORD

- Per la maggior parte degli usi NON è mai necessario avere la password in chiaro
- Usare funzioni di hash
  - una funzione non invertibile
  - da stringa di lunghezza arbitraria a lunghezza predefinita
  - Impossibile trovare una stringa dato un hash
  - Impossibile trovare due stringhe con lo stesso hash
- Usare hash crittograficamente sicuri:
  - bcrypt, PBKDF2

LA PASSWORD

- ➔ Non inventare un proprio schema di cifratura
- ➔ Codificare la password con un algoritmo di HASH (possibilmente lento)
- ➔ Usare un salt per evitare attacchi di tipo “rainbow table attack”

```
hash('fixedsalt:' + username + ':' + password)
hash(randomsalt + ':' + username + ':' + password)
```

LA PASSWORD

- Creazione dell'utente da parte dell'admin
  - Nessuno dovrebbe conoscere la password dell'utente (nemmeno il supporto tecnico)
- L'utente deve essere autonomo nel ripristino della password (e-mail o admin)
  - Se qualcuno è in grado di inviare la password all'utente c'è qualcosa che non va

demo time

AUTORIZZAZIONE

- Una volta avvenuta l'autenticazione come gestiamo i privilegi?
  - Configurazione stateless?
  - Stato sul server?
  - Stato sul database?
  - Stato sul client...?!?!

JWT - JSON WEB TOKEN

- Permette di mantenere in modo sicuro lo stato dell'applicazione sul client
- Il server è completamente stateless
- Il server ha modo di capire se lo stato è stato manomesso
- Lo stato può essere cifrato

JWT - JSON Web Token

```
# Il token un header il payload e la firma

header = '{"alg":"HS256","typ":"JWT"}'

# Le specifiche raccomandano un timestamp (iat)

payload = '{"user":"admin","iat":1422779638}'

# La firma viene calcolata con la base64 di key, header e payload

key           = 'secretkey'
unsignedToken = base64(header) + '.' + base64(payload)
signature     = HMAC-SHA256(key, unsignedToken)

# Infine viene calcolato il token e inviato al client

token = base64(header) + '.' + base64(payload) + '.' +
base64(signature)

# Il cliente deve inoltrarlo al server con l'header Authorization
```

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

✓ Signature Verified

SHARE JWT

DelphiDay italian conference

OAUTH2

- Scopo
  - Accesso alle API di svariati servizi (Google, Facebook, Twitter, Instagram, LinkedIn...)
  - Autenticazione (SSO)
- Tipologie principali
  - Authorization Code: usata per accedere a delle risorse con l'autorizzazione esplicita dell'utente
  - Client Credentials: usate per accedere a delle risorse server to server

OAUTH2

- Scopo
  - Accesso alle API di svariati servizi (Google, Facebook, Twitter, Instagram, LinkedIn...)
  - Autenticazione (SSO)
- Tipologie principali
  - Authorization Code: usata per accedere a delle risorse con l'autorizzazione esplicita dell'utente
  - Client Credentials: usate per accedere a delle risorse server to server

OAuth2 – 1/3

Step 1: ottenere l'authorization code

```
https://oauth2server.com/auth?response_type=code&client_id=ID&
  redirect_uri=REDIRECT_URI&scope=photos&state=kfjg893
```

response_type=code - Indica che l'app richiede un codice di autorizzazione
client_id - ID creato in fase di registrazione
redirect_uri - Url da richiamare dopo l'autenticazione
scope - a quali informazioni è interessata l'app
state - stringa casuale verificata in seguito

```
https://client.com/callback?code=12345&state=kfjg893
```

code - è il codice di autenticazione
state - deve essere lo stesso inviato dal client

OAuth2 – 2/3

```
# Step 2: ottenere il token

POST https://oauth2server.com/token?
  grant_type=authorization_code&code=AUTH_CODE&
  redirect_uri=REDIRECT_URI&
  client_id=CLIENT_ID&client_secret=CLIENT_SECRET

# grant_type - Tipo di autenticazione accordata (potrebbe essere
anche password, in questo caso viene saltato il passaggio
precedente e qui vengono inviati utente e password)
# client_id, client_secret - concordata col provider oauth
# code - il codice ricevuto in precedenza
# redirect_uri, scope, state - lo stesso di prima
```

OAuth2 – 3/3

```
# Se va tutto bene il server restituisce un json:

{
  "Access_token": "RsT5OjbzRn430zqMLgV3Ia",
  "Expires_in": 3600
  "Token_type":"bearer",
  "Scope":"read",
    "Info":{
      "name":"Rossi Mario",
      "email":"mario@email.com"
    }
}

# In caso di errore

{
  "Error": "invalid_request"
}
```

Scalabilità

3

STATELESS

- L'applicazione non conserva uno stato:
  - Nessuna sessione per utente
  - Nessuna informazione salvata tra una chiamata all'altra
- Dove mettere le informazioni che servono:
  - Inviate ad ogni chiamata
  - Sul database
  - Key/value store (Redis)
  - Dati sensibili sul client?... JWT

IDEMPOTENZA

- Indica un'operazione che, se anche eseguita più volte, non cambia il risultato
- Utile in caso si voglia tentare di rieseguire un'operazione il cui esito è incerto
- In ReST le operazioni idempotenti sono:
  - GET
  - PUT
  - [DELETE]

LOAD BALANCING

- Benefici:
  - Fault tolerance
  - Prestazioni
- Ce ne sono varie tipologie:
  - Hardware appliance
  - Software: mod_proxy, HAProxy, ...

LOAD BALANCING

- Sviluppo in delphi:
  - Configurazione della porta
  - Configurazione del nome del servizio
  - Path per log e file generati dall'applicazione

demo time

DATABASE

- Il database ha un ruolo fondamentale
  - Replica: consente di avere più copie dei dati sincronizzate in tempo reale o a fronte di un evento (master/slave, muti-master)
  - Sharding: consente di partizionare il database su più macchine

CACHING

- Se fatto bene può eliminare numerose chiamate al DB (e non solo)
- Sul client
  - Il client salva la sua copia dei dati
  - Il server dà una stima di durata della cache
- Sul proxy
  - Il proxy può fornire cache per servizi diversi
- Sul server
  - Il Può usare una cache locale (Redis, Memcache)

Thread safety

4

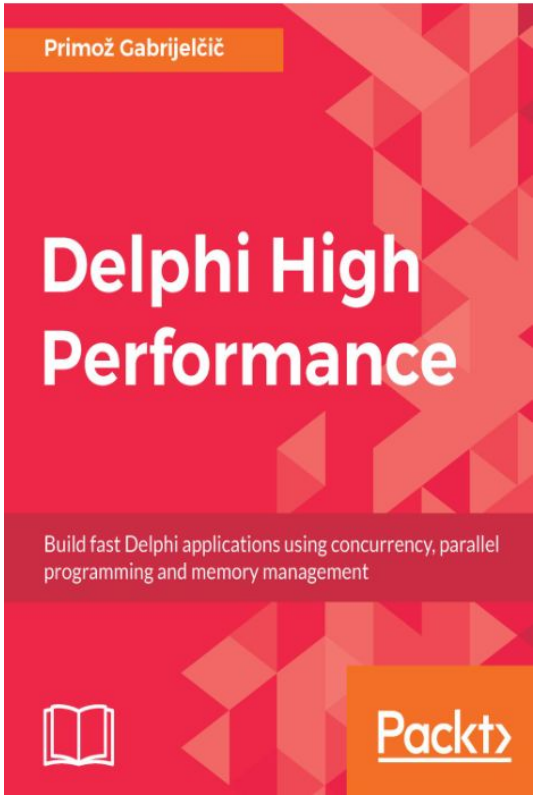
DelphiDay italian conference

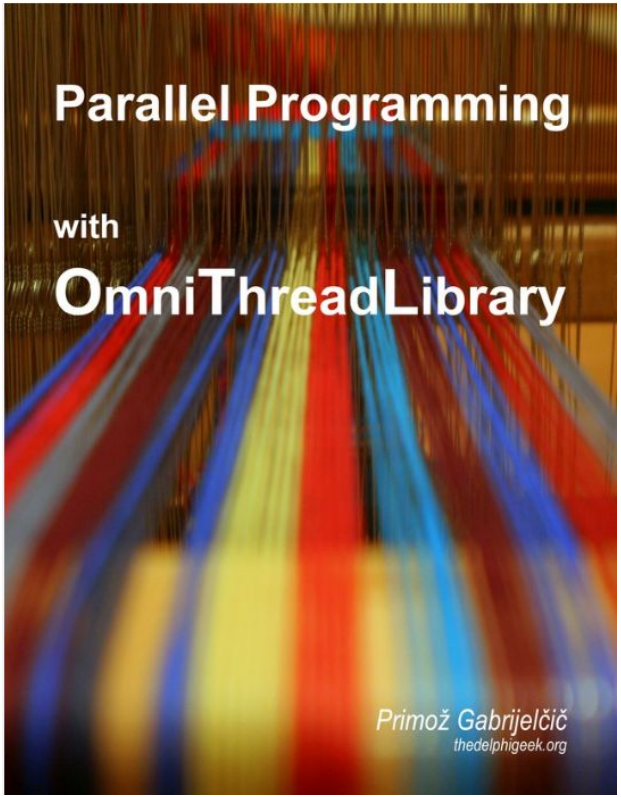
THREAD

- Ogni richiesta viene gestita da un thread specifico
- La stessa richiesta può essere più volte contemporaneamente
- WiRL crea un'istanza specifica della risorsa per ogni richiesta/thread, quindi:
  - Le variabili di istanza della richiesta sono sicure
  - Tutto il resto deve essere valutato (variabili globali, connessioni, accesso ai file)

THREAD


Copyrighted Material
Delphi
THREAD SAFETY PATTERNS
Dalija Prasnikar
Neven Prasnikar Jr.
Copyrighted Material


Primož Gabrijelčič
Delphi High Performance
Build fast Delphi applications using concurrency, parallel programming and memory management
Packt


Parallel Programming
with
OmniThreadLibrary
Primož Gabrijelčič
thedelphigeek.org

VARIABILI

- Mantenere la maggior parte delle informazioni sulla risorsa
- Variabili globali/sigleton: possono essere usati se sono readonly (attenzione al lazy loading)
- Il resto deve essere protetto
  - CriticalSection
  - Operazioni atomiche
  - Oggetti thread-safe

Thread-safe (1/4)

```
// “Lock” è un oggetto di tipo TCriticalSection. Deve essere lo stesso
per tutte le chiamate


Lock.Acquire;
try
  // Accesso alla risorsa non sicura (variabile, file, …)
finally
  Lock.Release;
end;
```

Thread-safe (2/4)

```
// Lazy loading thread-safe


function GetCustomer: TCustomer;
begin
  Lock.Acquire;
  try
    if not Assigned(Customer) then // Customer è una variabile globale
      Customer := TCustomer.Create;
    Result := Customer;
  finally
    Lock.Release;
  end;
end;
```

Thread-safe (3/4)

```
// Lazy loading thread-safe (double-check)

function GetCustomer: TCustomer;
begin
  if Assigned(Customer) then
  begin
    exit(Customer);
  end;

  Lock.Acquire;
  try
    if not Assigned(Customer) then
      Customer := TCustomer.Create;
    Result := Customer;
  finally
    Lock.Release;
  end;
end;
```

Thread-safe (4/4)

```
// Lazy loading thread-safe (compare and swap). Attenzione: in alcuni casi può creare due
istanze dell'oggetto customer, ma se si accorge che non serve la elimina

function GetCustomer;
var
  LCustomer: TCustomer;
begin
  if not Assigned(Customer) then
  begin
    LCustomer := TCustomer.Create;
    if InterlockedCompareExchangePointer(Pointer(Customer), LCustomer, nil) <> nil then
      LCustomer.Free;
  end;
  Result := Customer;
end;
```

demo time

DATABASE

- La maggior parte dei database gestiscono la concorrenza senza particolari problemi
- Ma in delphi la maggior parte dei componenti di accesso al DB non sono thread-safe
- La cosa più sicura e manterne una copia per thread
- In caso di problemi di performance valutare pooling

demo time

FILESYSTEM

- Se due thread accedono allo stesso file può succedere di tutto
- Quindi a seconda dell'uso è possibile:
  - Creare file univoci per thread (file temporanei)
  - Usare lock di qualche tipo (log, esportazione di dati)
  - Usare librerie thread-safe

E molto altro

5

DelphiDay italian conference

ALTRO

- Gestione della cache lato server
- Suddivisione del lavoro in più processi (PIPE, TCP/IP)
  - Permessi, performance, bug, ...
- Gestione operazioni lente
- Utilizzo dell'hardware specifico di una macchina non disponibile dal browser (stampante, scanner)

THANK YOU

DelphiDay
italian conference