Low-Code Software Security

Miguel Pupo Correia

ISCTE-IUL Low-code Software Development Summer School ‘2018
Motivation: bad software

• NASA Mars Climate Orbiter
  – $165 million
  – Crashed due to a units conversion bug

• NASA Mars Pathfinder
  – $265 million
  – Stopped for several hours due to a priority-inversion bug
Motivation: May 12, 2017

Empresas e bancos alvos de ataque informático
Na PT, trabalhadores receberam ordem para desligar as máquinas e e ser mandados para casa. Veja a mensagem recebida pelos trabalhadores.
PT
TVI24.OL.PT

Ataque informático. O que foi, como se espalhou, quem o travou
Um poderoso vírus entrou por uma falha do Windows e alastrou na rede. Criou o caos em hospitais e empresas de todo o mundo.
OBSEVADOR.PT

Portugal Telecom alvo de ataque informático internacional
A Portugal Telecom é um dos alvos do ataque informático que afetou várias empresas em Portugal, Espanha e Alemanha. A espanhola Telefónica é outra...
OBSEVADOR.PT

NHS hit by massive ransomware attack, many hospitals and clinics offline
The ransomware attack appears to be spreading to more NHS trusts.
ARSTECHNICA.CO.UK

Ataque informático mundial: empresas portuguesas afetadas
Vírus ataca apenas os utilizadores que tenham sistema operativo da Microsoft
DN.PT | POR DIÁRIO DE NOTÍCIAS
Motivation: 2017 in numbers

• **Coin mining** [cryptojacking] was the biggest growth area
• **Ransomware** infections are up 40 percent in 2017, driven primarily by WannaCry
• 1 in 13 **URLs** analyzed at the gateway were found to be **malicious**. In 2016 this number was 1 in 20
• 62 percent increase in overall **botnet activity**
• **zero-day vulnerabilities** recorded in 2017: 4262
• new discovered **mobile malware** variants grew 54%
• 24,000 **malicious mobile applications** blocked per day
Motivation: last week (!)

https://www.facebook.com/seginfportugal/
Motivation: low code vs cloud

- **Low code platforms** have much in common with **cloud computing**, so also similar security threats:
  - Data breaches
  - Data loss
  - Account hijacking
  - Insecure APIs
  - Malicious insiders
  - Shared technology issues
  - …
Outline

Security concepts
Low-code software security problem
  Users and basic protections
  Web vulnerabilities and protections
  Mobile vulnerabilities and protections
Low-code software development life cycle
  Platform security
Wrap-up
SECURITY CONCEPTS
What is security?

- **Confidentiality** – absence of disclosure of data by non-authorized parties
- **Integrity** – absence of invalid system or data modifications by non-authorized parties
- **Availability** – readiness of the system to provide its service

- “non-authorized” requires a **security policy**, explicit or implicit
Why is security needed?

• **Direct economic impact** – security violation impacts business operation (loss of systems or data)
• **Indirect economic impact** – loss of reputation
• **Human / environment impact** – may kill people, cause pollution, etc.
• **Compliance** – legislation requires security, e.g., GDPR, NIS directive

• …**life&death issues**, for companies and even people
Vulnerabilities

• **Vulnerability** – a system (hw/sw) defect that may be exploited by an attacker to subvert security policy

• They are **defects** but some **developers** don’t think so:
  – “the team leaders conveniently assumed that security vulnerabilities were not defects and could be deferred for future enhancements or projects.”

• **0-day vulnerability** – a vulnerability not publicly known, only privately
Types of software vulnerabilities

• **Design vulnerability**
  – inserted during the software design

• **Coding vulnerability**
  – introduced during coding (often a bug with security implications)

• **Operational vulnerability**
  – caused by the software configuration or the environment in which it is executed
Attacks

- **Attack** – action(s) done with the intent of activating a vulnerability

Source: OWASP top 10
• **CWE** – Common Weakness Enumeration
  – A taxonomy of vulnerabilities - http://cwe.mitre.org/

• **CVE** – Common Vulnerabilities and Exposures
  – A catalog of vulnerabilities - http://cve.mitre.org/
  – Also as **NVD** – National Vulnerability Database

• **CAPEC** – Common Attack Pattern Enumeration and Classification
  – A taxonomy of attacks - https://capec.mitre.org/
Attack surface

• **Attack surface** – interfaces from which attacks come
  – 1\textsuperscript{st} question when speaking of an application security: what’s the attack surface?
  – not trivial to understand in large software
Attacks

• Can be **interactive** or **autonomous** (with malware)
• Can be **technical** vs. **social engineering**
• Can be **directed** or not
Risk

Objective is not to achieve 100% security but to have an acceptable risk (why?)

Probability of successful attack = Threat level x Vulnerability level

Risk = Probability of successful attack x Impact
LOW-CODE SOFTWARE SECURITY PROBLEM
Low-code software architecture

Internet/Network

Users

Backend/Cloud

Data store

Application server

Developers ("low code")
Architecture – not radically new

MS Windows
Mac OS X, Linux

Android
iOS,...

client – server
Security – not radically new

Internet/Network

Backend/Cloud
Data store
Application server
Cloud security
Platform security

Users
Web app security
Mobile app security
Software development security

Developers ("low code")
Outline

• Security concepts
• Low-code software security problem
  • Users and basic protections \(\rightarrow\) what’s already there
  • Web vulnerabilities and protections \(\rightarrow\) up to you
  • Mobile vulnerabilities and protections \(\rightarrow\) up to you
  • Low-code software development life cycle \(\rightarrow\) up to you
• Platform security \(\rightarrow\) up to you / platform provider
• Wrap-up
USERS AND BASIC PROTECTIONS
User Authentication

• **Participants** = \{developers, users\}

• **Authentication** – showing to the server (in this case) that it’s me who is trying to access
  – Binding of identity to a **subject** (a computer entity)

• **Common approaches**
  – username / password
  – 2-factor authentication: add SMS, smartcard, biometry,…
  – **Single sign-on**: same authentication for accessing several systems
Access Control

• **Access control** – restrict who can do what
  – Participants have **permissions**; can do operations if they have the corresponding permission
  – Examples (for low code platform): permission to list applications, deploy applications, full control

• **Common approaches**
  – **Access control lists** – for each service/object there’s a list of which subjects can do what
  – **Role-based access control** – permissions assigned to roles, roles assigned to subjects
Example creating roles

Create a role:

Assign permissions to a role:

Source: OutSystems
Communication security

• Client-server protection using HTTPS (SSL/TLS)
  – Authenticates server using public-key crypto (certificates)
  – Protects confidentiality by encrypting communication
  – Protects message integrity/authenticity by adding message authentication codes

• REST API
  – Leverages HTTPS security
  – Major issue is user authentication – schemes seen before can be used (username/password, etc.)
All set!

- Only authorized users
- They can only do what they are allowed to
- Communications are secured

Secure?
WEB VULNERABILITIES AND PROTECTIONS
• Client-server model
• Original: static HTML pages sent over HTTP; stateless
• Today: higher layer protocols (HTTPS, REST); server-side and client-side code; stateful
Don’t trust input!
A1: Injection

- Main case: **SQL Injection**

- Example (PHP/MySQL):

  ```php
  $username = $HTTP_POST_VARS['username'];
  $password = $HTTP_POST_VARS['passwd'];
  $query = "SELECT * FROM logintable WHERE user = " . $username . " AND pass = " . $password . ""
  $result = mysql_query($query);
  if(!$result) die_bad_login();
  ```

  username: root
  password: root’ OR pass <> ‘root

  Query: `SELECT * FROM logintable WHERE user = 'root' AND pass = 'root' OR pass <> 'root'`
A1: Injection

• There are several forms (SQL, XML, LDAP, XPath, XSLT, HTML, OS command injection, …)

• All have in common:
  – Attacks come from inputs (don’t trust inputs)
  – There is some server-side interpreter (e.g., DMBS, LDAP)
  – Applications accepts metadata in inputs (e.g., ‘ ’)

• Protection:
  – Use a safe API (parameterized statements) – best
  – Accept only known-good inputs (whitelisting)
  – Sanitize/encode inputs, e.g., with EncodedSQL()
A2: Broken Authentication and Session Management

• Several issues:
  – User credentials are unprotected, guessable, or modifiable
  – Session IDs are exposed / fixable
  – Authentication not invalidated with logout

• Example: session ID in the url (trivial to ride the session)
  – http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

• Protection:
  – follow checklist of best practices
A3: Cross Site Scripting (XSS)

- Allows attacker to run script in users’ browsers
- Stored XSS:

1. Store script
2. Get and run script

Internet/Network

Web clients aka browsers

Application server

Storage

Data store
A3: Cross Site Scripting (XSS)

- Reflected XSS:

1. email with script
2. request with script
3. get and run script
A3: Cross Site Scripting (XSS)

• Protection:
  – Input whitelisting
  – Input sanitization with reliable libraries
  – Output encoding with reliable libraries, e.g.,
    EncodeJavascript(), EncodeHTML()
A4: Insecure Direct Object Reference

• Vulnerability: site exposes a reference to an internal object and no proper access control
  – Object ex.: file, directory, database record, key (URL, form parameter)
  – The attacker can manipulate these references to access other objects without authorization

• Ex.: direct reference to file in web page:
  – <select name="language"><option value="fr">Francais</option>
  – Embeds file fr.php but attacker may send otherfile

• Protection:
  – Don’t expose refs (use session info), proper access control
A5 / A9: Security Misconfiguration, Components with Known Vulnerabilities

• Several issues:
  – Vulnerable/out of date software: OS, server, DBMS, libraries
  – Unnecessary/dangerous features enabled/installed
  – Default accounts
  – Security settings not properly set

• Protections:
  – Configure properly (hardening)
  – Check for software updates automatically
  – Run vulnerability scanners
A6: Sensitive Data Exposure

• Several issues:
  – Sensitive data not encrypted, encrypted with unsafe algorithms (e.g., home-made, DES), or weak keys
  – Hard-coding keys and storing keys in unprotected stores

• Protections:
  – Use strong algorithms and keys, considering the threats
  – Store keys securely
A7: Missing Function Level Access Control

• Users access private or privileged functionality
  – e.g., pages are not protected, just inaccessible from the normal web tree (security by obscurity)
  – Attack: forced browsing

• Protection:
  – Proper access control
  – No “hidden” pages as form of protection
A8: Cross-Site Request Forgery (CSRF)

1- link with operation in email or webpage

0- ongoing session

Web clients aka browsers

2- attacker’s operation

Data store

Application server
A8: Cross-Site Request Forgery (CSRF)

• Protection:
  – Insert large nonce as a hidden field in the form; do not accept operation if nonce doesn’t come
  – Critical actions: re-authenticate
A10: Unvalidated Redirects and Forwards

- Used to trick victims into malicious websites
  - Example: site has a page called `redirect.jsp` which takes a single parameter named `url`
  - Attacker crafts a good-looking URL that redirects users: `http://www.nicepage.com/redirect.jsp?url=evil.com`

- Prevention:
  - Avoid redirects/forwards; avoid using inputs in them; validate inputs
  - Use functions that replace domain in the URL with your domain: `ReplaceURLDomain()`
MOBILE VULNERABILITIES AND PROTECTIONS
Mobile

- **Devices:**
  - smartphones, tablets
- **Operating systems:**
  - Android, iOS,...
- **Applications:**
  - typically webapps but client is an app, not a browser

## Architecture

<table>
<thead>
<tr>
<th>Apps (phone, contacts, browser,... built in and loaded from store)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application framework / services (windows, notifications, resources, location,...)</td>
</tr>
<tr>
<td>Runtime (Android: ART/Dalvik ~JVM)</td>
</tr>
<tr>
<td>Kernel (Android: based on Linux; iOS based on Darwin/BSD)</td>
</tr>
<tr>
<td>Hardware (usual + RF transceiver, SIM card, NFC, GPS, sensors,...)</td>
</tr>
</tbody>
</table>
Low-code software security

Focus: complex environment

Internet/Network

Backend/Cloud

Data store

Application server

Developers ("low code")

Users

= web security ➔ done!
Security problems

• Users download many apps from marketplaces, some of which are malicious
  – Google Play Store, Apple App Store, Aptoide, etc., etc.
  – Apps claim permissions, users typically grant them
  – Bad apps may do attacks by themselves (e.g., steal data) or tamper with behavior of legitimate apps

• Personal/critical data stored in devices

• Unsecure network access (e.g., open wifi)
OWASP Top Ten Mobile Risks

- There’s a 2016 edition, but more a classification than a top 10
- Not showing all, but those farther away from the web top 10

<table>
<thead>
<tr>
<th>Year</th>
<th>Mobile Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014</td>
<td>M1: Weak Server Side Controls</td>
</tr>
<tr>
<td></td>
<td>M2: Insecure Data Storage</td>
</tr>
<tr>
<td></td>
<td>M3: Insufficient Transport Layer Protection</td>
</tr>
<tr>
<td></td>
<td>M4: Unintended Data Leakage</td>
</tr>
<tr>
<td></td>
<td>M5: Poor Authorization and Authentication</td>
</tr>
<tr>
<td></td>
<td>M6: Broken Cryptography</td>
</tr>
<tr>
<td></td>
<td>M7: Client Side Injection</td>
</tr>
<tr>
<td></td>
<td>M8: Security Decisions Via Untrusted Inputs</td>
</tr>
<tr>
<td></td>
<td>M9: Improper Session Handling</td>
</tr>
<tr>
<td></td>
<td>M10: Lack of Binary Protections</td>
</tr>
</tbody>
</table>
M2: Insecure Data Storage

- Developers assume that users or malware can’t access stored data, so they don’t protect it
  - Storage places: SQLite databases, SD card, cloud synced, log files, property list / XML / manifest files
  - Relevant data: usernames, passwords, cookies, personal information, app data

- Protection:
  - Encrypt stored data (use proper libraries)
  - Enforce access control, e.g., not MODE_WORLD_READABLE in Android
M3: Insecure Authentication

• Weak authentication allows adversary to do arbitrary operations in the app or backend
  – Weak authentication is prevalent due to mobile devices’ input form factor (promotes PINs/short passwords)
  – Users often offline, so offline authentication may be allowed and it’s insecure (hard: malicious host threat)

• Protection:
  – Assume offline authentication can be bypassed, so re-authenticate with the backend when online
  – Do local integrity checks to detect unauthorized changes
M7: Client Side Injection

• Code injection in the mobile app (instead of in the backend), typically in apps using browser libraries
  – Variants of XSS and local SQL injection (in SQLite)
  – New: abusing phone dialer + SMS, abusing in-app payments

• Protection:
  – Parameterized queries; disable JavaScript; etc.
M10: Lack of Binary Protections

• Lack of protections against reverse engineering
  – Allow stealing confidential data, fraud, piracy, intellectual property theft
  – Several attack tools available: ClutchMod (cracker for iOS), dex2jar (Android), IDA Pro, Hopper (disassembler), gdb
  – Malicious host problem: not entirely solvable

• Protection:
  – Detect jailbreak and debuggers; use checksums; etc.
Secure?
LOW-CODE SOFTWARE DEVELOPMENT LIFE CYCLE
Security Development Lifecycle

• The term is generic, but the best known SDLC is Microsoft’s – for normal software development:

• What shall we do for low code development?
Low-Code Security Development Lifecycle

- Provide software security training for *low code* developers
  - “at least one security training class each year” MS SDL 5.2
• Define the security requirements; some sources:
  – Specific project business requirements, misuse cases
  – Legislation (e.g., GDPR, NIS directive)
  – Standards (e.g., ISO/IEC 27034 Application security, IEEE 1012-2012 Software Verification and Validation)
  – Microsoft SDL 5.2 (for this and all the next ones)
Low-Code Security Development Lifecycle

- **Best practices**
  - e.g., CSD “Avoiding the top 10 software security design flaws”, OWASP Top 10s, low code platform vendor docs
- **Threat modeling**
  - Non-trivial but very useful if application is complex
- **Security design principles**
  - Keep design simple, least privilege, defense in depth,…
Low-Code Security Development Lifecycle

- Best practices, e.g., OWASP Top 10s, low code platform specific
- Static analysis tools – low code platform specific
  – may be integrated with IDE
- Enable dynamic low code platform specific protections if available
Low-Code Security Development Lifecycle

- Dynamic / fuzz testing
- Vulnerability scanners
- Tests based on the threat model (if available)
- Best practices, e.g., OWASP Testing Guide v4 or low code platform specific
Low-Code Security Development Lifecycle

- Final security review
  - e.g., peer or external code review
- Plan for when vulnerabilities are discovered (not if…)
  - patches, reports
- Plan for rollback to previous version
- Issue platform security recommendations
  - e.g., recommend Mobile Device Management (MDM)
Low-Code Security Development Lifecycle

- Collect information about security events, issue reports and patches
- Possibly run a Computer Security Incident Response Team (CSIRT) 24x7
PLATFORM SECURITY
Low-code software architecture

- Users
- Internet/Network
- Backend/Cloud
  - Data store
  - Application server
- Developers ("low code")
- Platform security
Running the platform

• on premises **versus** at provider/cloud
  – if at provider/cloud:
Platform protection — examples

• Virtual private networks / virtual LANs / firewalls
  — for communication security, traffic segregation, and filtering
• Anti-malware / IDS / IPS
  — for malware / attack detection and reaction
• Vulnerability management of the platform software
  — awareness of critical vulnerabilities, install updates
• Security Information and Event Management system
  — integrated security management (monitoring and control)
Platform protection – cloud example

Source: OutSystems Sentry datasheet
WRAP-UP
Conclusions

• Low code platform security is a new problem, but previous solutions mostly apply
  – Web security, mobile security, cloud security,…
• Focus on secure code implementation is important
• but developers must have a broad view of the secure software development life cycle
• Learn the best practices, employ the best tools
References


• OWASP documentation cited
• Microsoft SDL documentation cited
• OutSystems online security documentation
• Salesforce Security Guide and other Force.com docs
Thank you

Miguel Pupo Correia

http://www.gsd.inesc-id.pt/~mpc/