# TIE-23500, Web Software Development (Web-ohjelmointi)

## Notes by Daniel "3ICE" Berezvai <http://tut.3ice.hu/>

Security guest lecture.

We can't defend against hackerman (from Kung Fury).

But against not really good hackers, who are only motivated by money, we can. Or make it harder to hack than other websites. Not our problem anymore.

Involve the security guys in development. Not just penetration testing at the end.

## It's simple (But not easy)

Just do the right thing. Design it properly, do it right. Mistakes in implementation are bugs = security issues.

Expect the unthinkable.

You can also use bug bounties. (1000$ paid to any hacker who breaks in and safely discloses his methods.)

### Do the right thing

Don't roll your own. Especially don't reinvent hash algorithms and crypto. Not even your own random number generator. Just use a library.

Follow best practices. Don't invent your own.

Understand what you are doing. Read the RFC. Understand your tools and libs.

### Unthinkable

#### Names

; DROP TABLE "COMPANIES2 ; --LTD

This is an officially registered company.

There's also a polish company dropping users, and a security conference had a speaker from google named OR "some SQL"

#### Domains and DNS records

Domain looks exactly like microsoft.com but isn't.

Or also forward slash in subdomain exists. Even though it's illegal character.

(Puny code attack)

### SVG –JS demo

Arbitrary JavaScript execution… In a picture. XML file with svg extension?

This is unthinkable. Beware SVG images. (This reminds me of JPEG viruses from over 10 years ago.)

## Input sanitation

XSS, SQL injection, XXE, CSRF are all about input validation. (Cross site scripting, XML external entity attack, Cross site request forgery)

Simplest solution: White list what's allowed, deny everything else.

Session management can still fail, but if you don't invent your own, you'll be pretty safe.

## JavaScript necessary evil

Joke: Grep "evil" from js source code. Pages of results. Evil test, evil parameter, evil error.

## Demo/practice Gruyere

<https://google-gruyere.appspot.com/start>

Home cooked session management. Betrays information about user role, and possibly even their password. Makes forged requests easy.

Don't just whitelist elements, whitelist attributes as well. Otherwise you'll get pwned through event handlers. (onmouseover)

Stored XSS: Remains in the database.

Reflected XSS: No persistence, but we create a link that contains the attack code.

Gruyere has other modes. Upload files, edit your cookies, XMLHTTPRequest

SQL injection no included in this app.

We saw an SQL statement that creates C# binary in the database which can then be executed. So basically anything can happen if you achieve SQL injection.

In Oracle you can create, upload, and run Java class files. Connect to third party server and leak all the data from the db.

This is not standard ANSI SQL. But all big vendors have their own extensions. MySQL lets you select the contents of a file, for example.

## Even if you sanitize your input you are still failing

Logic attack: Survey – asks for a number 1-5 and the user can submit any integer, like 1000. Nothing fancy, not an injection. But skews the results on the report.

Penetration testing won't find this.

Enter negative price for a product. Win some prize money by submitting definitely winning answers outside the "allowed" range.

Billing system doesn't expect zero as a price.

Java has plenty of valid "integers" like INF, NaN. Don't put those in the database.

## Real world attack cases

User agent contains Shell shock code, which gets sent to CGI (){:;}; /bin/bash …

Payload was a Perl program opening IRC connection and joining botnet.

Attacker's name, password was in the source code. (Some Italian guy)

He didn't even write it himself, just copied from someone else "script kiddie"

## Who is responsible for the server? Dev or Ops?

Do you need to care as a developer? #DEVSEC

What is your responsibility?

## Some recent fails

### Burden of legacy

MD5 can be broken now. 15 years ago it was the best practice. But not today…

Native code is dangerous…

C++ buffer overflow possible even with Address space layer randomization / DEP protecting you.

Hard to sell to customer. "Pay me again to fix what I wrote 15 years ago"

### Shortcuts and anarchy

Heavy process, not understood / accepted by developers.

Domain is unauthorized. This system is a third party app.

Serving Upgrade.exe installation for the whole. Hosted on unauthorized server.

Google "how to exploit Application name + version"

Pwned in 15 minutes.

If they are running old versions, or third party things, google for known exploits.

### I accidentally…

Someone accidentally put credentials on Github and committed. This happens.

They promptly removed it, but git history still contains the old commit.

They didn't even change the passwords.

Even IBM did this

### Webhack event

Light weight, fun bug bounty hunt.

SQL injection → dumped the whole database.

"But our code was fine! What?"

#### One does not simply inject into…

Issue turned out to be a 0-day in Spring libraries.

Even if you do everything right, you can still fail.

Details: <https://github.com/solita/sqli-poc>

The end.

Teacher was running WiFi Pineapple on us during the lecture. Logged 200 new SSIDs.

As our devices look for "known" networks, searching broadcasts what's being looked for.

In active mode the device can even pretend to be those networks. (Illegal in Finland.)

It can even force disconnect us from our normal network.