XSS Countermeasures in Grails

Rafael Luque @rafael_luque — **OSOCO**
José San Leandro @rydnr — **Ventura24**

Groovy Madrid
Groovy Users Group

GRAILS by Pivotal

http://goo.gl/UGdJ0l

Create Account | Login

Search on grails.org

Home | Learn | Products & Services | Community | Downloads | Plugins

Plugins You can find out about all the publicly available Grails plugins.

<iframe width="90

<iframe width="90

FILTERS CLEAR FILTER

- All Plugins
- Featured
- Top Installed
- Popular
- Recently Updated
- Newest
- Supported by SpringSource
- Pending Plugins

POPULAR TAGS

- Javascript
- Ajax
- Security
- Persistence
- Database
- Functionality

Nothing matched your query -

Oh no, Grails.org was PWN3D!!

Don't panic! This is a simple reflected XSS used as a proof of concept in our talk about Grails and XSS prevention.

XSS Intro

XSS concepts

- What's a XSS
- XSS Types: Reflected, stored, DOM-based.
- Famous XSS attacks: Samy worm, MrBean defacement, ...

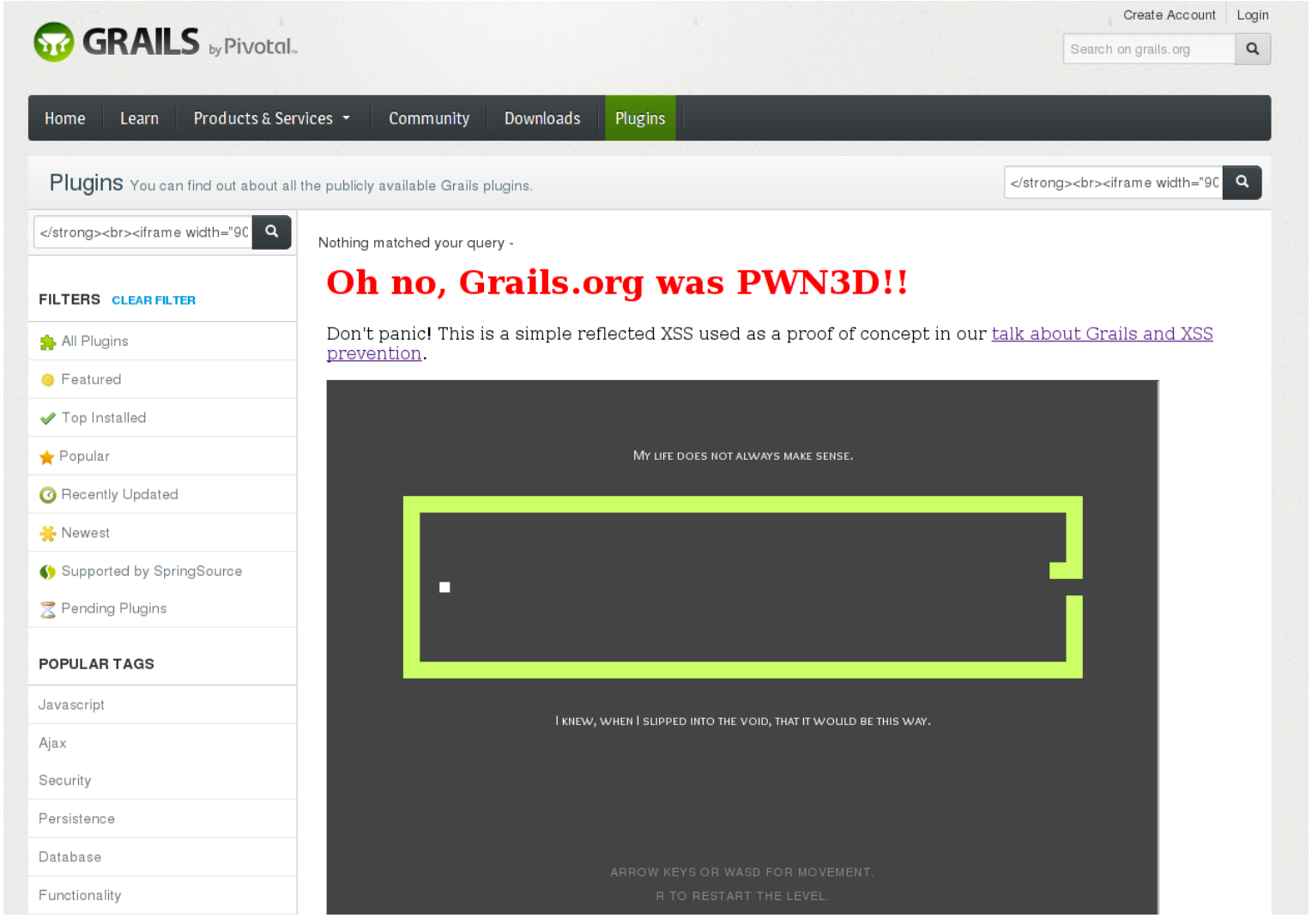
XSS threats

- Interface defacement
- Session hijacking
- Click hijacking
- Malware infection
- Your PC may be joined to the horde of zombies in a BotNet.

Following the white rabbit...

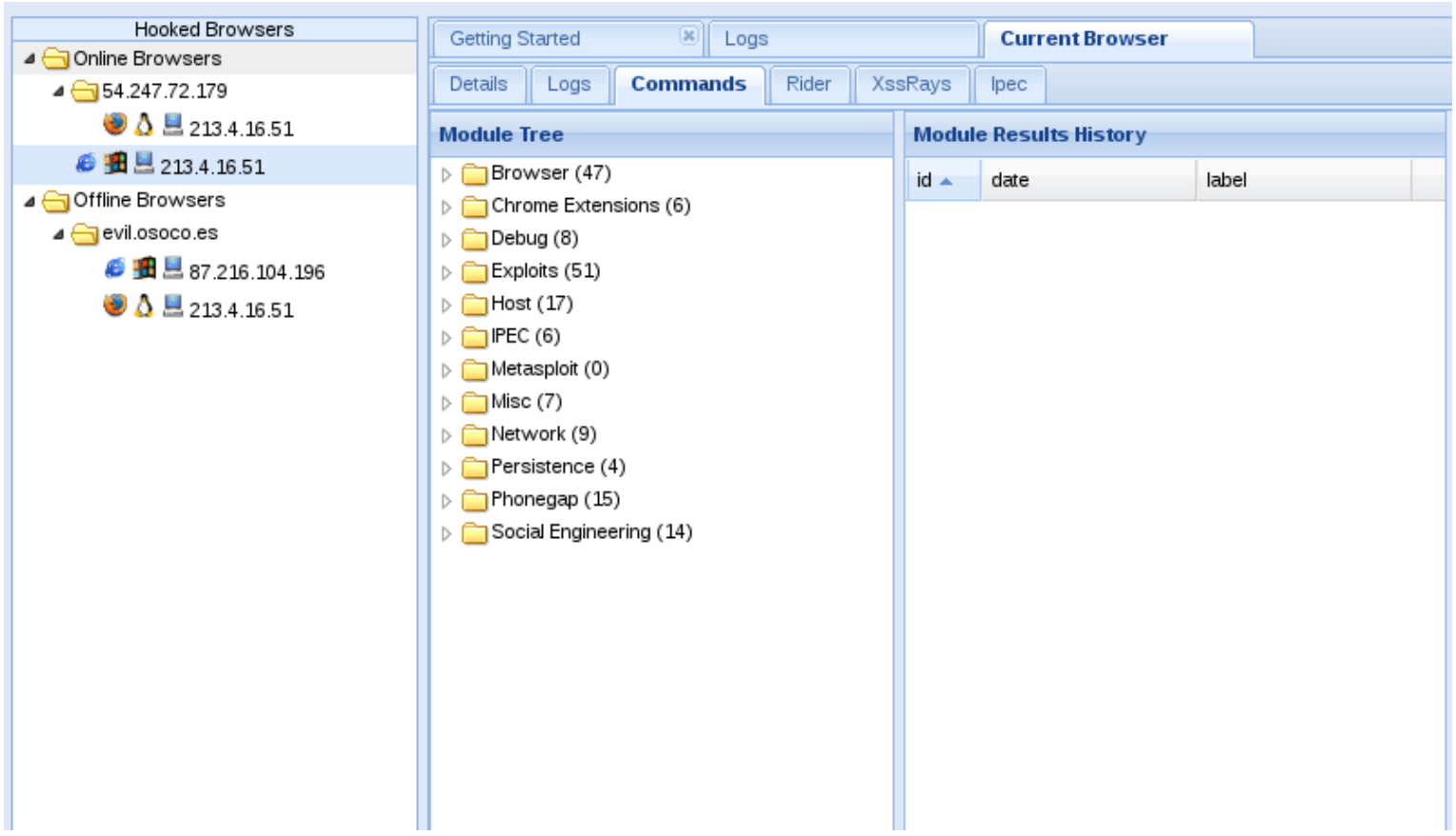
he white rab

Something more than a joke…



Hooking your browser

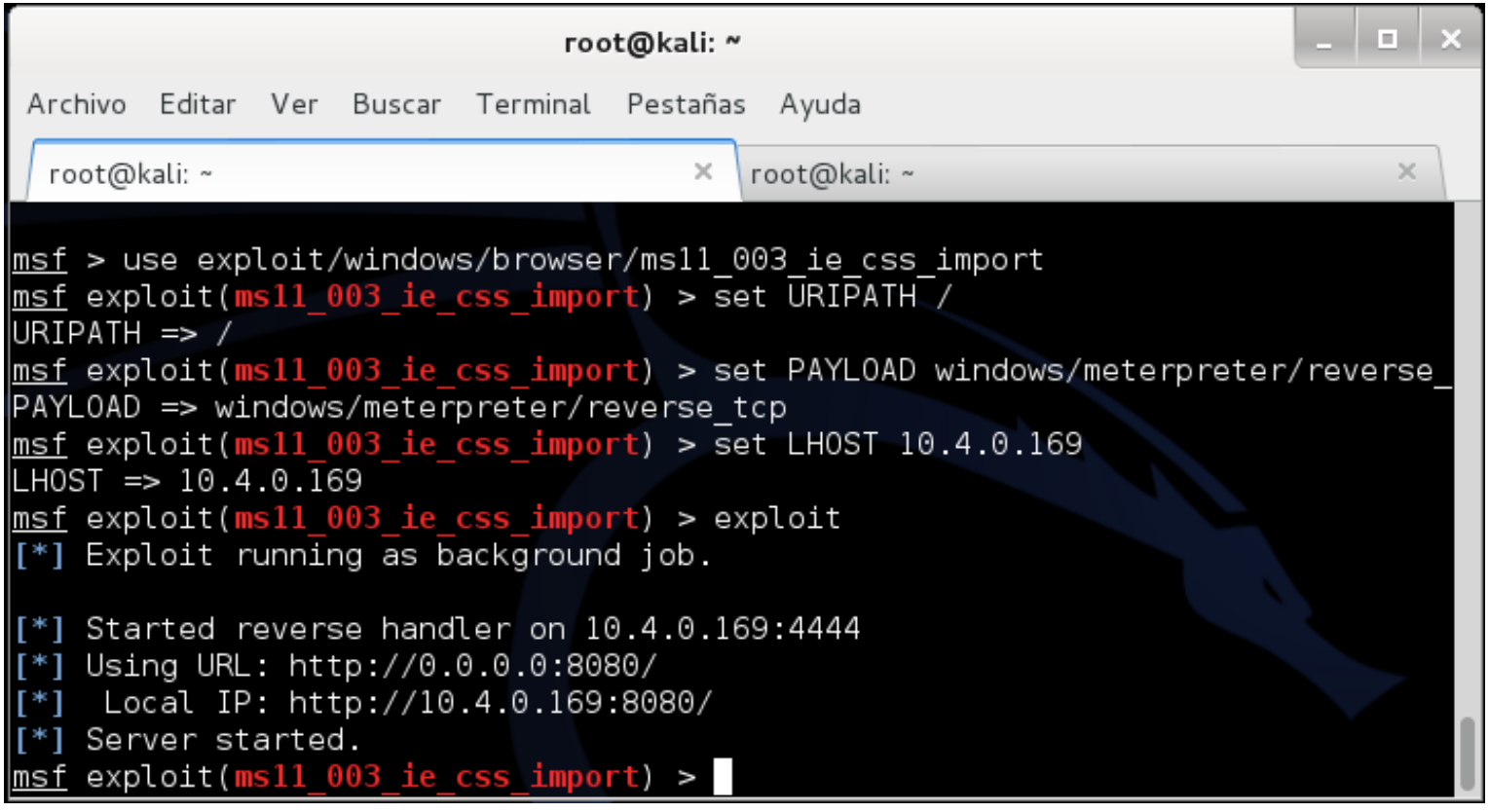
Hooked browsers with BeEF



Exploiting your system

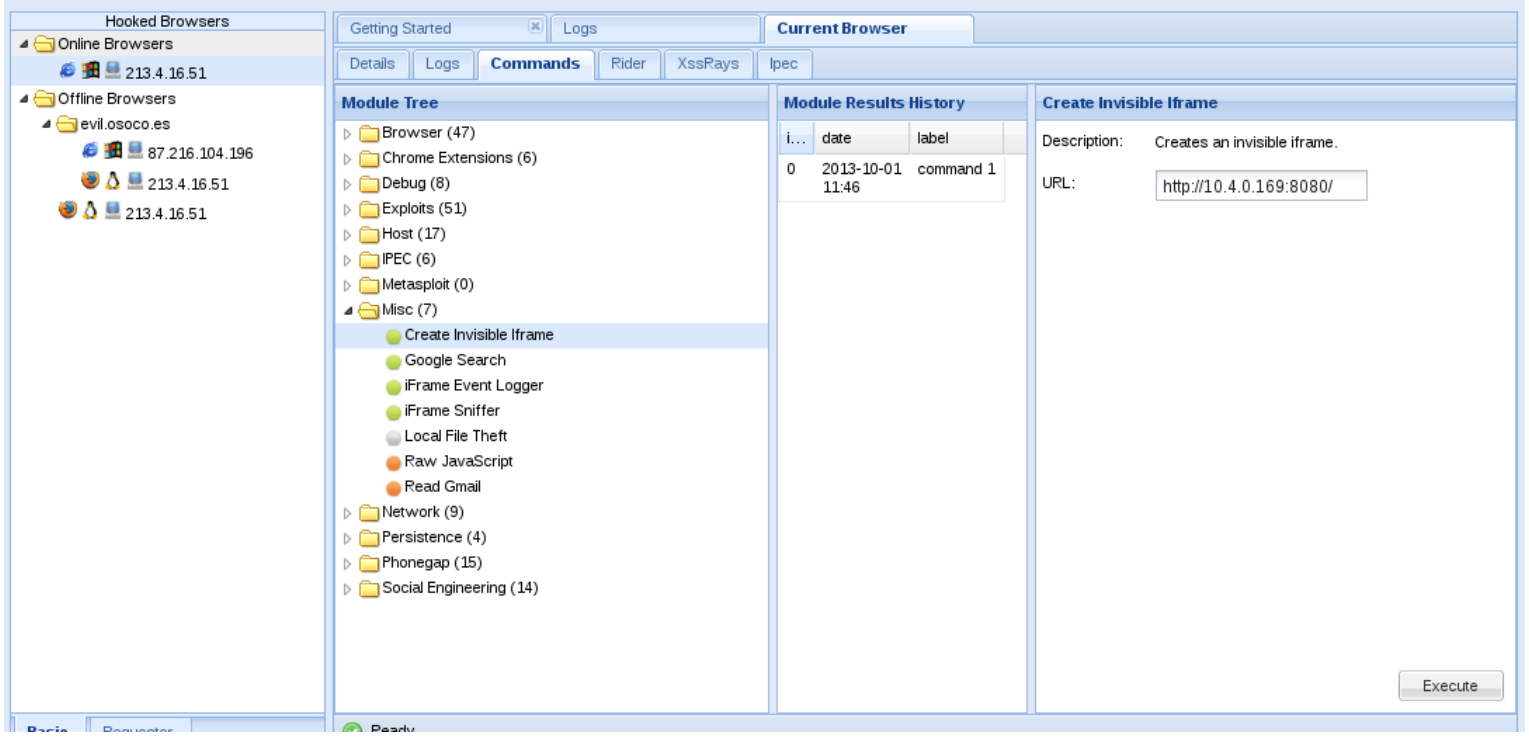
Exploiting the browser

1. Preparing the exploit server…



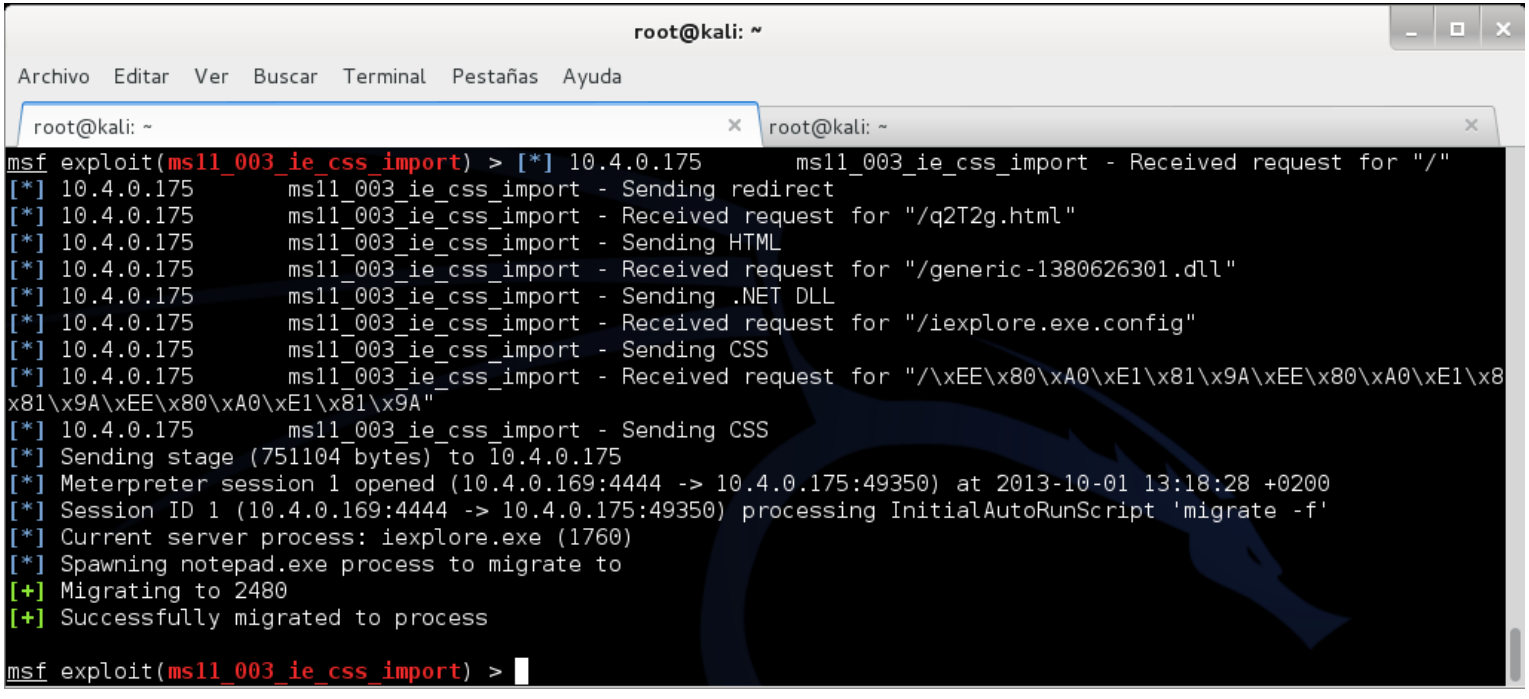
Exploiting the browser

2. Injecting an invisible frame pointing to the exploit server...



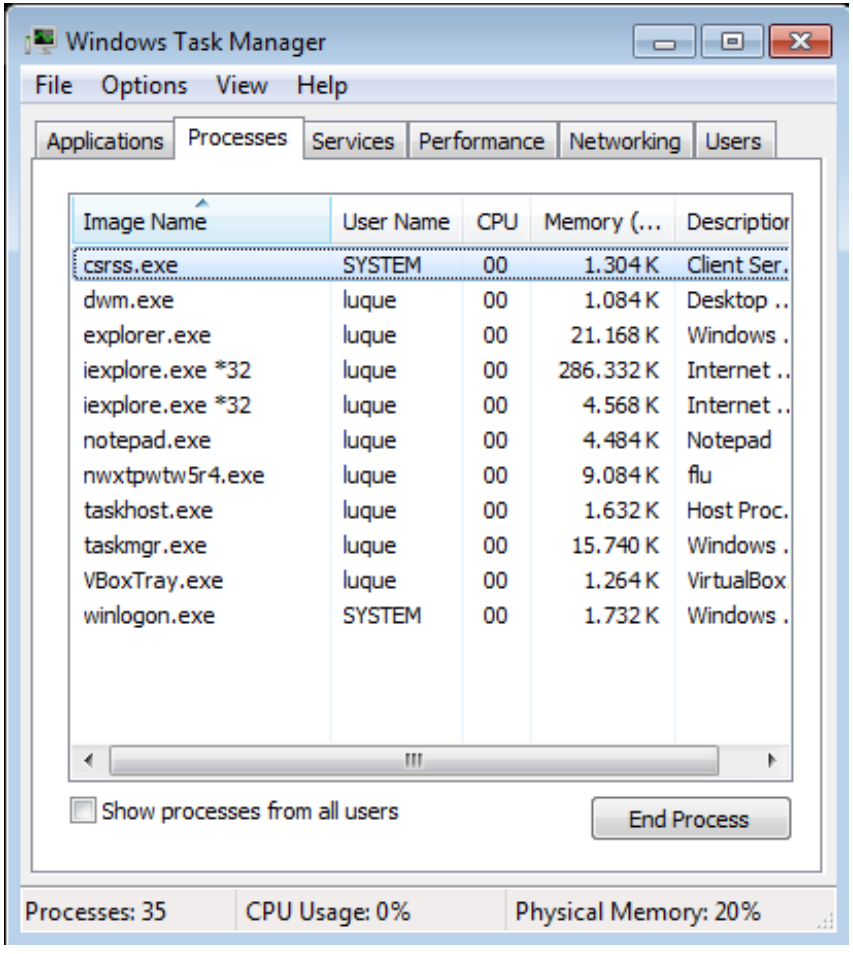
Exploiting the browser

3. Exploit works and executes the payload...



Exploiting the browser

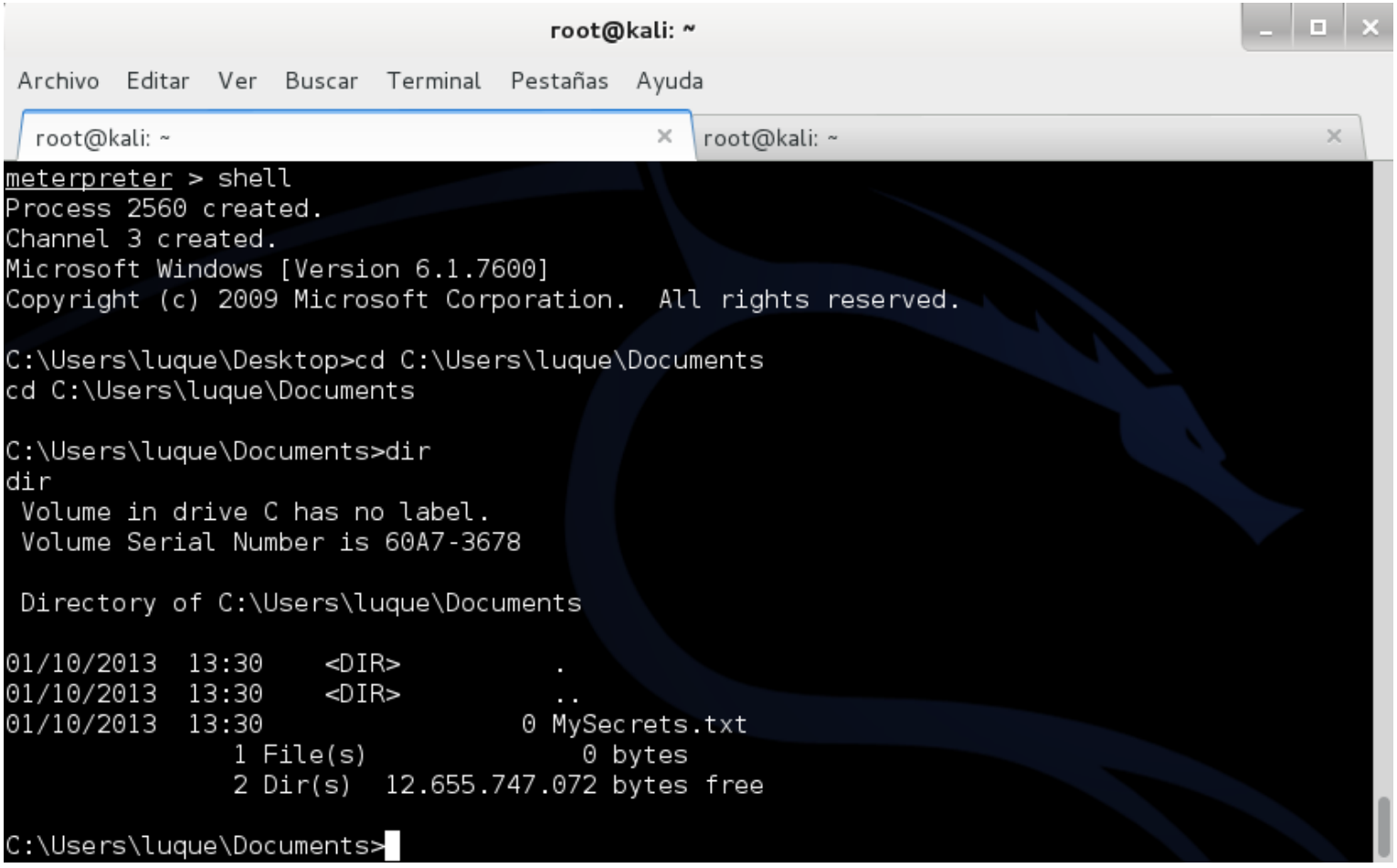
4. Spawning notepad.exe process to migrate to…



Fun with post-exploitation

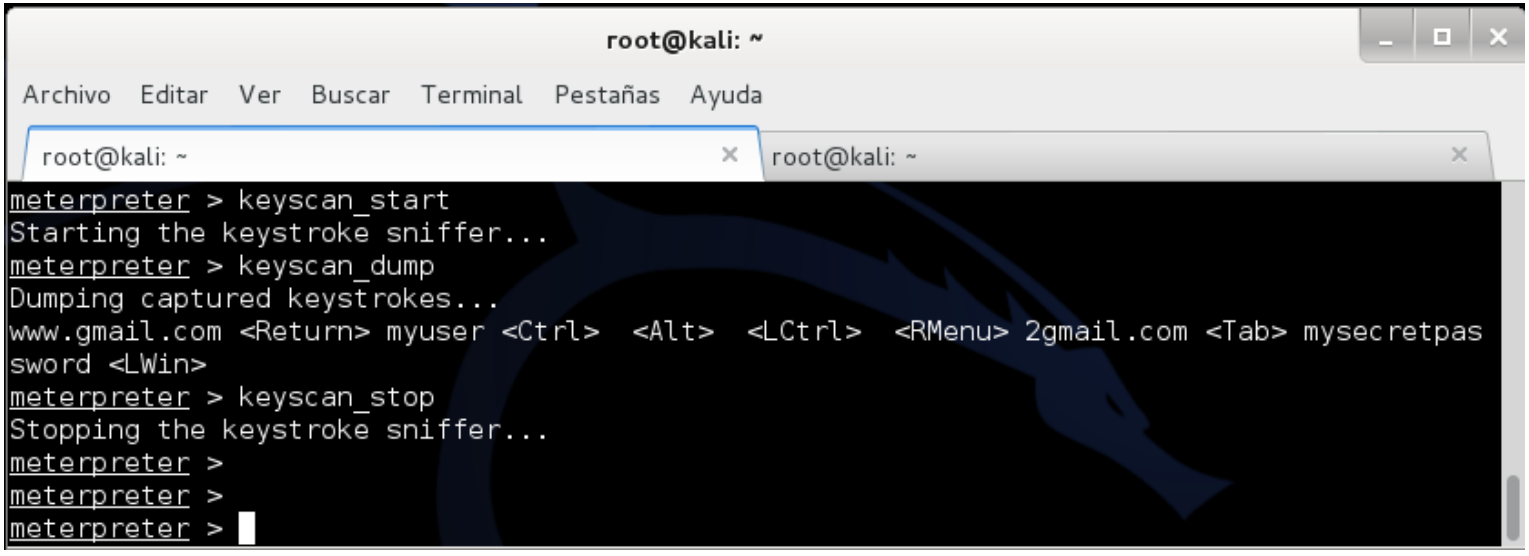
Post-exploitation phase

Run a remote shell



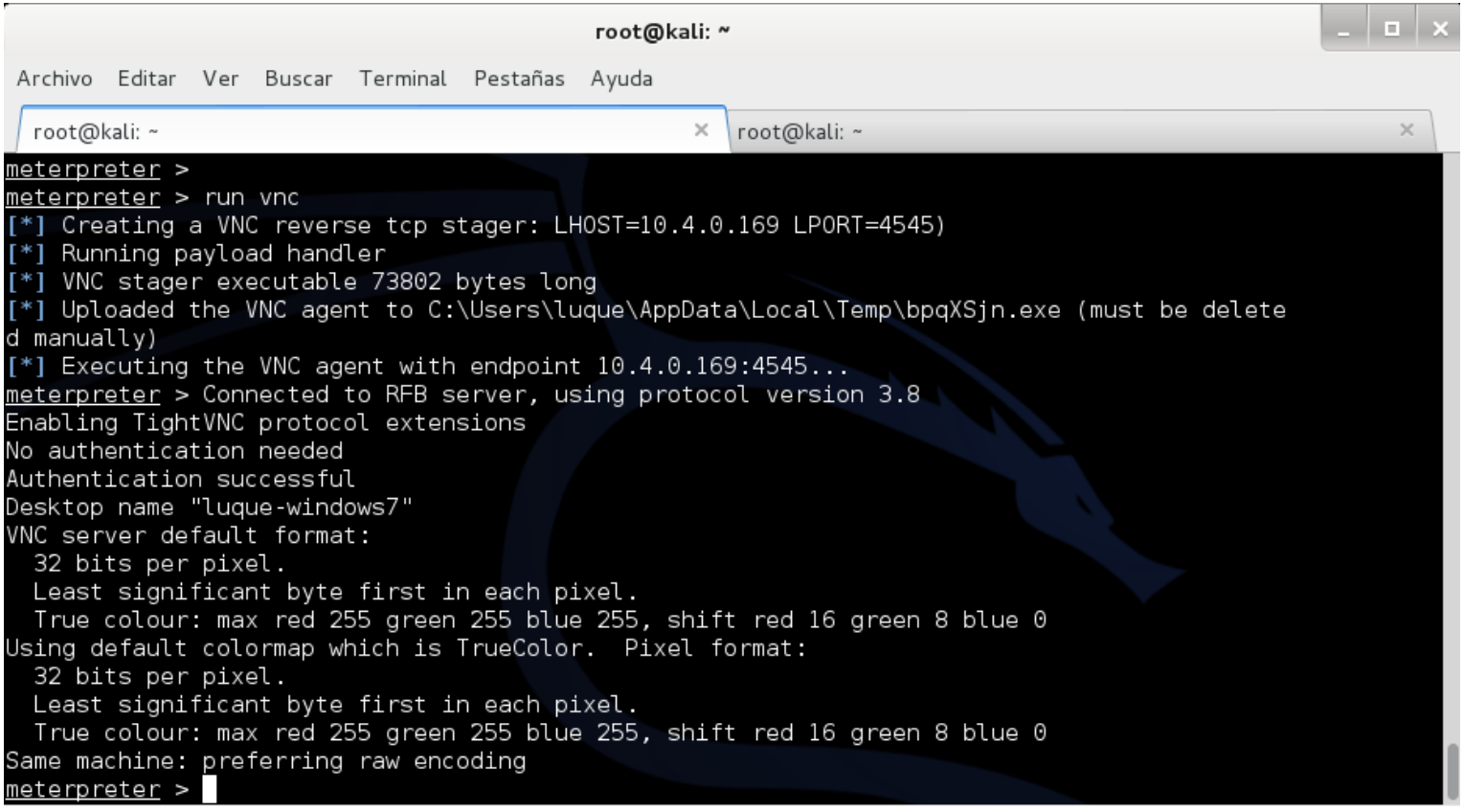
Post-exploitation phase

Keylogging



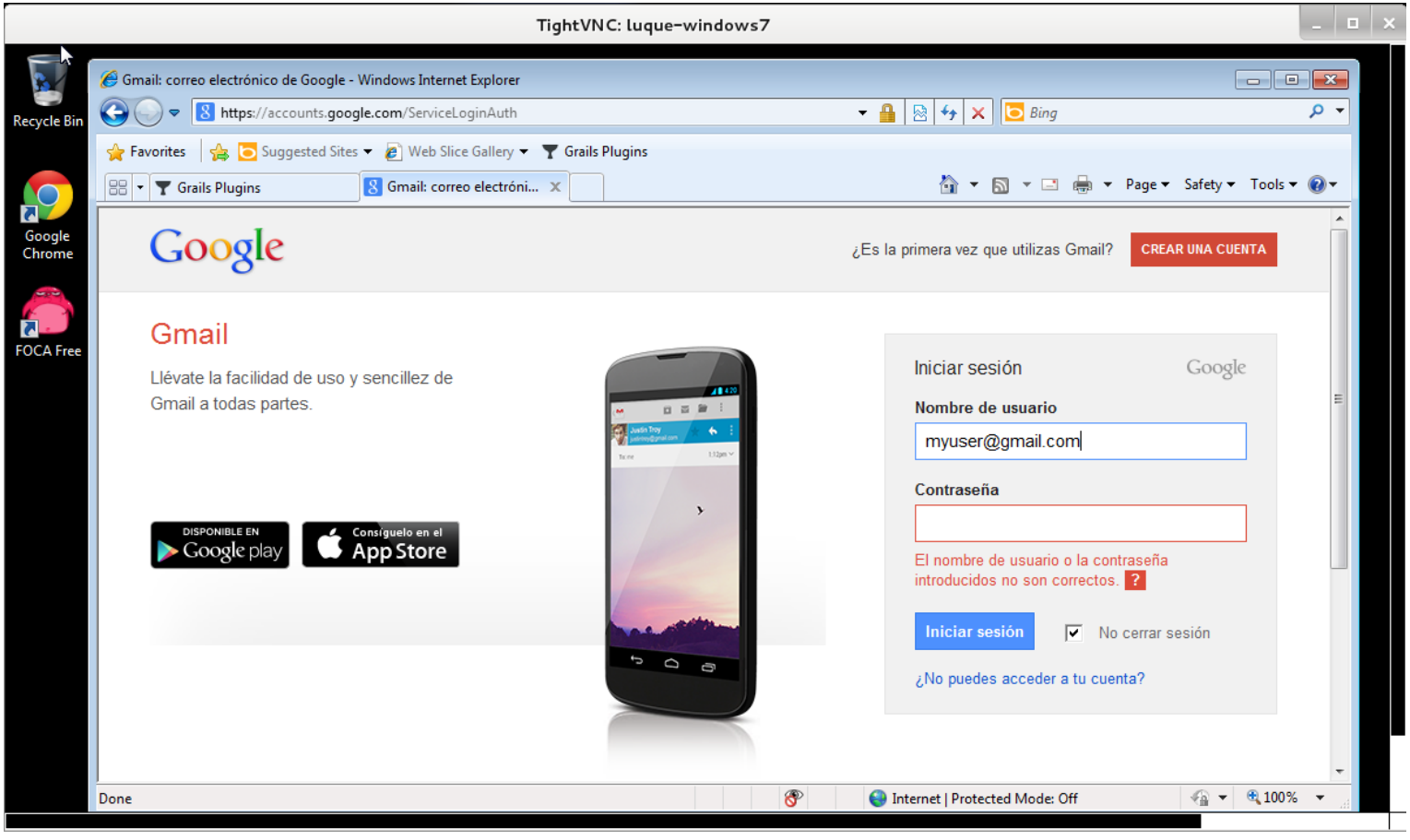
Post-exploitation phase

Run VNC session



Post-exploitation phase

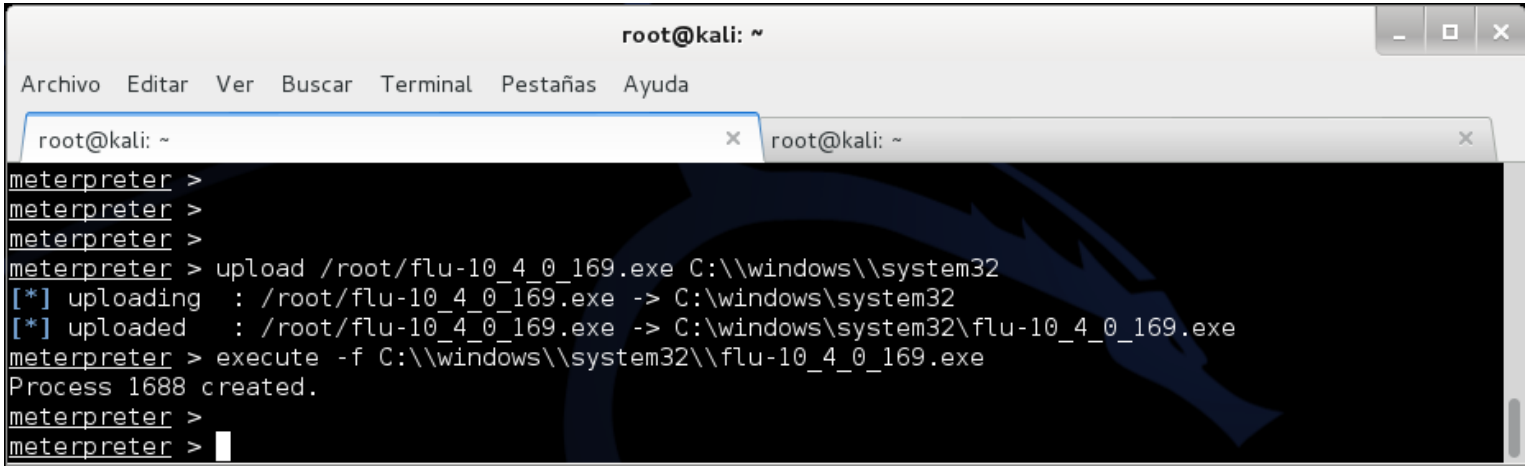
Run VNC session




Welcome to the horde of zombies

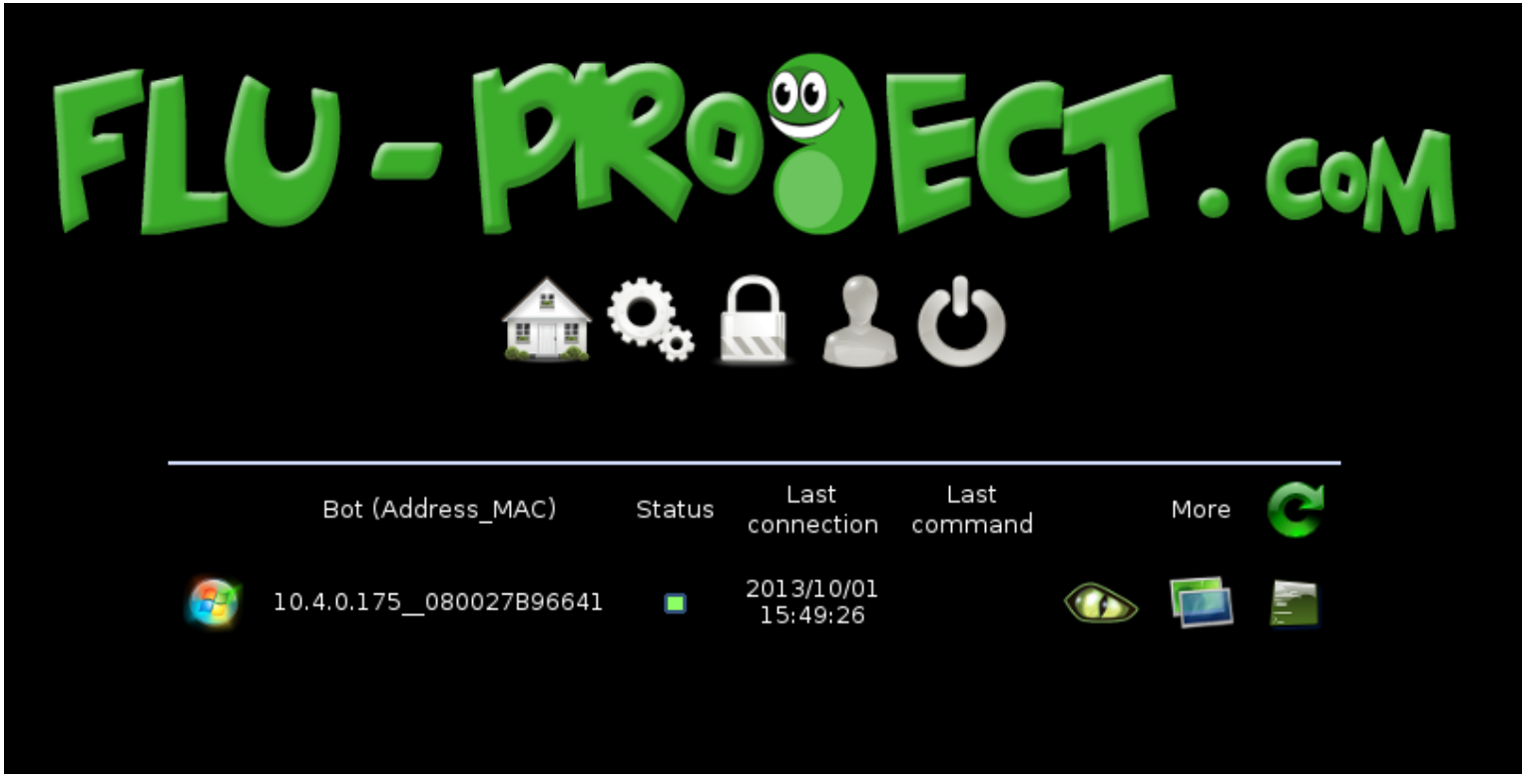
Joining to a botnet

1. Install the malware…



Joining to a botnet

2. Welcome to my botnet C&C. . .



Responsibilities: Why is this still an issue?

Commercial software

- XSS is not known for business stakeholders

Commercial software

- XSS is not known for business stakeholders
- For most people, security means attacking your servers

Commercial software

- XSS is not known for business stakeholders
- For most people, security means attacking your servers
- Developers don’t pay enough attention

Do your homework

- Raise awareness

Do your homework

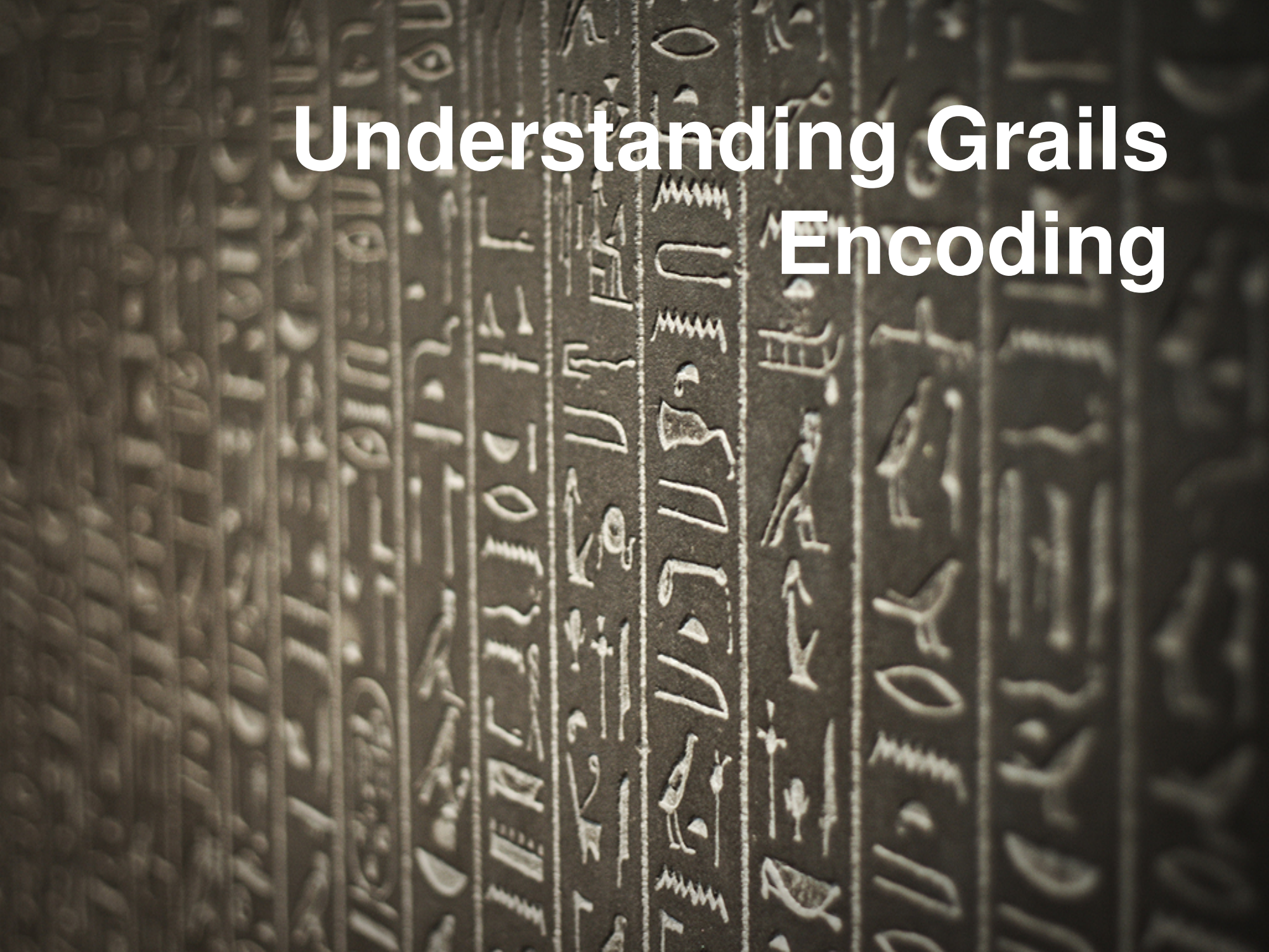
- Raise awareness
- Practice with security tools

Do your homework

- Raise awareness
- Practice with security tools
- Promote defensive coding

Do your homework

- Raise awareness
- Practice with security tools
- Promote defensive coding
- Improve monitoring



Understanding Grails Encoding

Grails Pre-2.3 Gotchas

#1: Built-in default codec

#1: Built-in default codec

```
grails.views.default.codec
```

#1: Built-in default codec *is none!*

```
grails.views.default.codec = ''none''
```

#1: Built-in default codec *is none!*

Problems

You have to escape explicitly every untrusted data:

```
encodeAsHTML()
encodeAsJavaScript()
encodeAsURL()
```

#1: Built-in default codec *is none!*

Problems

High likelihood of XSS vulnerabilities in production.

E.g. Grails.org website.

#1: Built-in default codec *is none!*

Problems

Double-encoding prevention over *Security by default*.

#1: Built-in default codec *is none!*

Solution

Change default codec to HTML:

```
grails.views.default.codec = ''html''
```

#2: Inconsistent behaviour

Apply codec

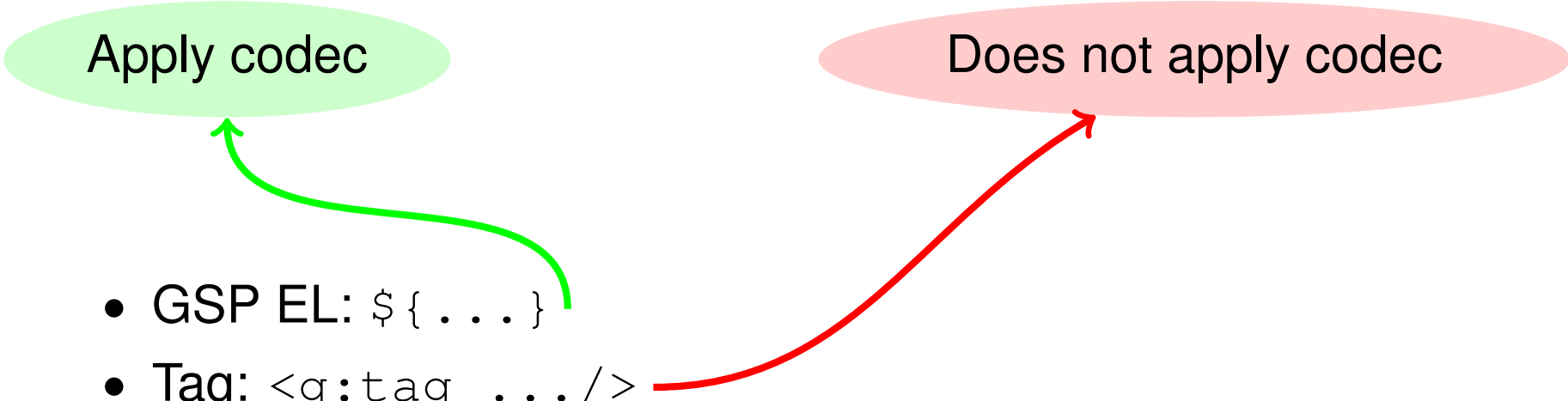
Does not apply codec

- GSP EL: `${...}`

#2: Inconsistent behaviour

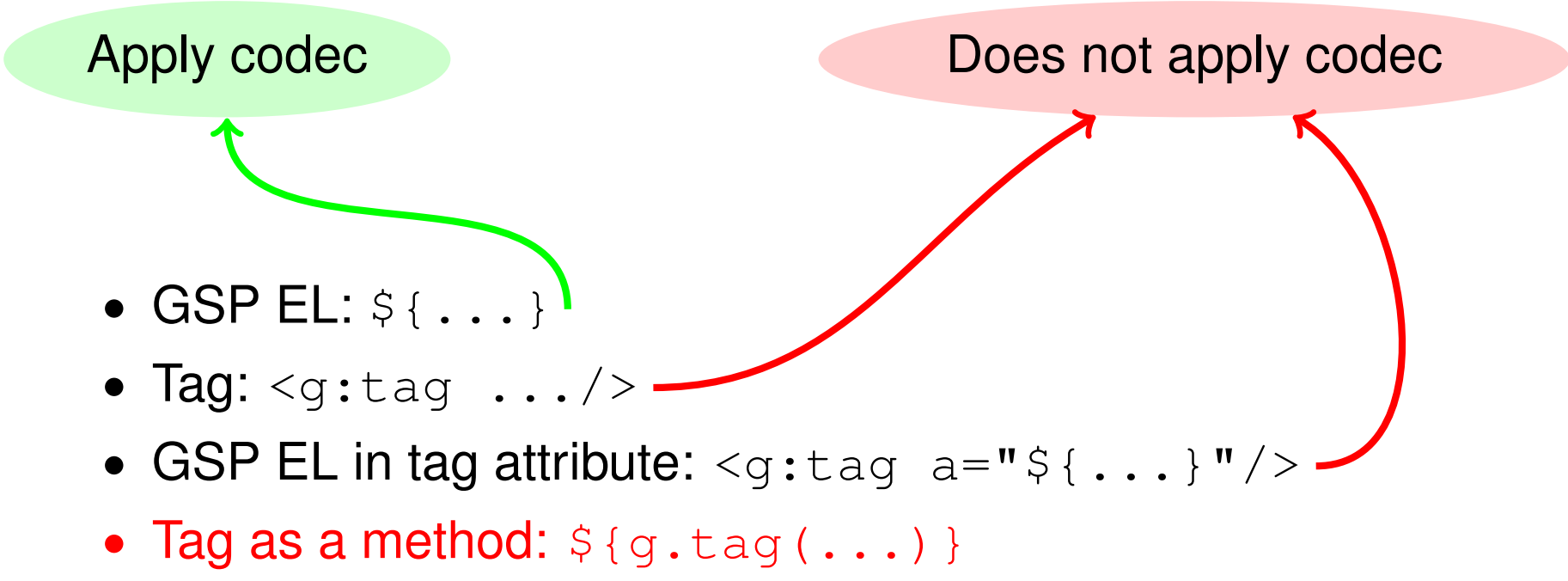
- GSP EL: `${...}`
- Tag: `<g:tag .../>`

#2: Inconsistent behaviour

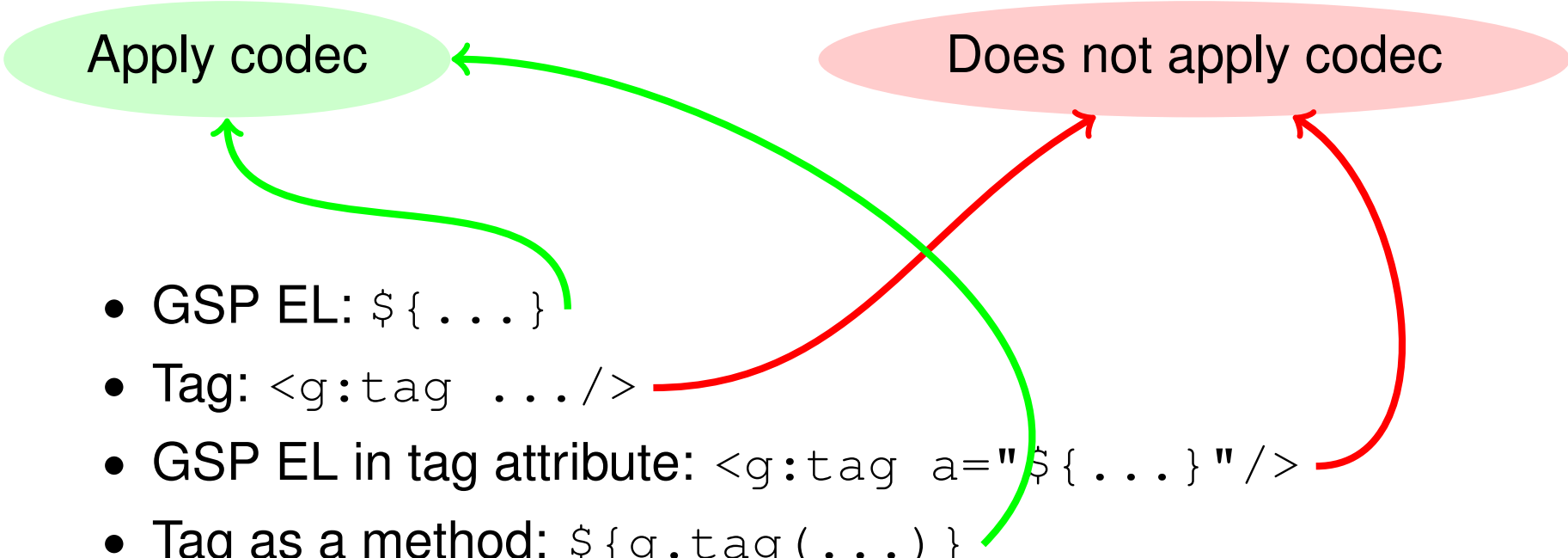


- GSP EL: `${...}`
- Tag: `<g:tag .../>`
- GSP EL in tag attribute: `<g:tag a="${...}"/>`

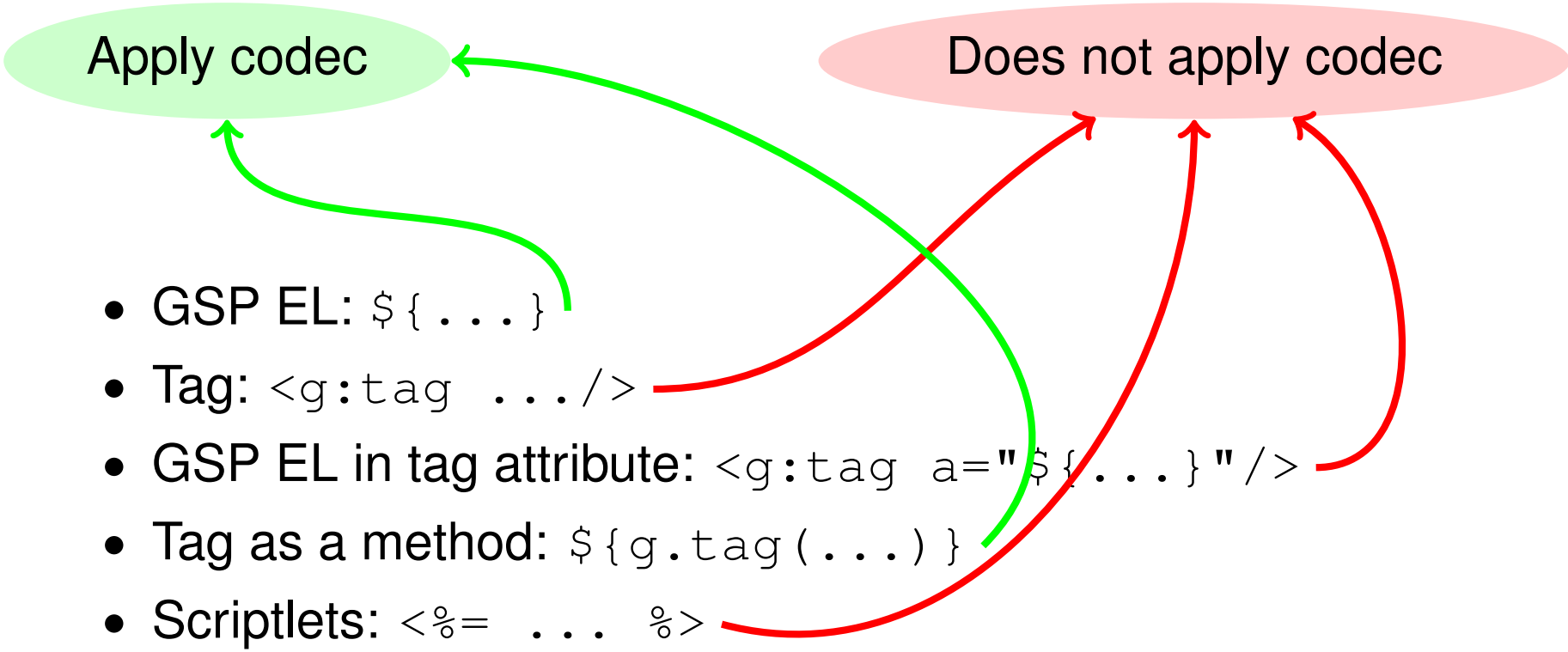
#2: Inconsistent behaviour



#2: Inconsistent behaviour



#2: Inconsistent behaviour



#3: Tag output is not escaped

Problems

Review the tags you use to make sure they encode their output or have options for this (e.g. `encodeAs` attribute).

#3: Tag output is not escaped

Problems

Review the tags from plugins you use.

#3: Tag output is not escaped

Problems

Review the tags you invoke as methods in Controllers.

#3: Tag output is not escaped

Problems

Don’t trust Grails core tags, they have inconsistent behaviour. E.g:

```
<g:fieldValue /> // HTML-encoded
<g:message />    // NO HTML-encoded
```

#3: Tag output is not escaped

Solutions

If tag implementation doesn't encode, add it explicitly or invoke it as a method inside a GSP expression:

```
<g:message ... encodeAs=''HTML''/>
${g.message(...)}
g.message(...).encodeAsHTML()
```

#4: g:message doesn't escape arguments

Problems

With default codec set to HTML the following XSS attack vector works:

```
<g:message code='welcome' args='[params.user]'/>

where:
   welcome = Hi {0}!
   params.user = <script>alert('pwnd')</script>
```

#4: g:message doesn't escape arguments

Solutions

Upgrade to a Grails version with the issue (GRAILS-7170) fixed:

2.0.5, 2.1.5, 2.2.2, 2.3-M1

#4: g:message doesn't escape arguments

Solutions

Escape explicitly or invoke the tag inside a GSP expression:

```
<g:message code='welcome' args='[params.user]'
    encodeAs='HTML'/>

${g.message(code:'welcome', args:[params.user])}
```

#5: One codec is not enough

You MUST use the escape syntax for the context of the HTML document you're putting untrusted data into:

- HTML
- JavaScript
- URL
- CSS

#5: One codec is not enough

HTML entity encoding doesn't work if you're using untrusted data inside a <script>, or an event handler attribute like onmouseover, or inside CSS, or in a URL.

#5: One codec is not enough

Problems

You can override the default codec for a page, but not to switch the codec for each context:

```
<%@page defaultCodec='CODEC' %>
```

#5: One codec is not enough

Problems

How to manage GSPs with mixed encoding requirements?

#5: One codec is not enough

Solutions

Turn off default codec for that page and use `encodeAsJavaScript()` and `encodeAsHTML()` explicitly everywhere.

#5: One codec is not enough

Solutions

Extract the JavaScript fragment to a GSP tag encoding as JavaScript.

Grails 2.3 Encoding Enhancements

#1: New configuration more *secure by default*

#1: New configuration more security by default

```
grails {
  views {
    gsp {
      encoding = 'UTF-8'
      htmlcodec = 'xml' // use xml escaping instead of HTML4
      codecs {
        expression = 'html' // escapes values inside ${}
        scriptlet = 'html' // escapes output from scriptlets in GSPs
        taglib = 'none' // escapes output from taglibs
        staticparts = 'none' // escapes output from static templates
      }
    }
    // escapes all not-encoded output at final stage of outputting
    filteringCodecForContentType {
      //'text/html' = 'html'
    }
  }
}
```

#2: Finer-grained control of codecs

Control the codecs used per plugin:

```
pluginName.grails.views.gsp.codecs.expression = 'CODEC'
```

#2: Finer-grained control of codecs

Control the codecs used per page:

```
<%@ expressionCodec='CODEC' %>
```

#2: Finer-grained control of codecs

Control the default codec used by a tag library:

```
static defaultEncodeAs = 'HTML'
```

Or on a per tag basis:

```
static encodeAsForTags = [tagName: 'HTML']
```

#2: Finer-grained control of codecs

Add support for an optional `encodeAs` attribute to all tags automatically:

```
<my:tag arg='foo.bar' encodeAs='JavaScript'/>
```

#3: Context-sensitive encoding switching

Tag `withCodec('CODEC', Closure)` to switch the current default codec, pushing and popping a default codec stack.

```
out.println '<script type=''text/javascript''>'
withCodec(``JavaScript'') {
    out << body()
}
out.println()
out.println '</script>'
```

#3: Context-sensitive encoding switching

Core tags like `<g:javascript/>` and `<r:script/>` automatically set an appropriate codec.

#4: Raw output

When you do not wish to encode a value, you can use the `raw()` method.

```
${raw(book.title)}
```

It's available in GSPs, controllers and tag libraries.

#5: Default encoding for all output

You can configure Grails to encode all output at the end of a response.

#5: Default encoding for all output

```
grails {
  views {
    gsp {
      codecs {
        expression = 'html' // escapes values inside ${}
        scriptlet = 'html' // escapes output from scriptlets in GSPs
        taglib = 'none' // escapes output from taglibs
        staticparts = 'raw' // escapes output from static templates
      }
    }
    // escapes all not-encoded output at final stage of outputting
    filteringCodecForContentType {
      'text/html' = 'html'
    }
  }
}
```

Check your Plugins security

Plugins are also part of your application

- Grails plugins are not security audited

Plugins are also part of your application

- Grails plugins are not security audited
- Grails plugins are part of your application's attack surface

Plugins are also part of your application

- Grails plugins are not security audited
- Grails plugins are part of your application's attack surface
- Review plugins to make sure they encode, and if they don't you should JIRA the authors immediately, and fork and patch to fix your app quickly.

Demo: Javamelody vulnerability

- CVE-2013-4378 vulnerability reported.

Demo: Javamelody vulnerability

- CVE-2013-4378 vulnerability reported.
- Allows **blind XSS** attack via `X-Forwarded-For` header spoofing.

Demo: Javamelody vulnerability

- CVE-2013-4378 vulnerability reported.
- Allows **blind XSS** attack via `X-Forwarded-For` header spoofing.
- The attack target is the admin's browser.

Demo: Javamelody vulnerability

- CVE-2013-4378 vulnerability reported.
- Allows **blind XSS** attack via `X-Forwarded-For` header spoofing.
- The attack target is the admin's browser.
- Fixed in the last release (1.47).

Demo: Javamelody vulnerability

- CVE-2013-4378 vulnerability reported.
- Allows **blind XSS** attack via `X-Forwarded-For` header spoofing.
- The attack target is the admin's browser.
- Fixed in the last release (1.47).
- You should upgrade ASAP.

Solutions: What options do we have?

Think like an attacker

- According to your grails version

Think like an attacker

- According to your grails version
- Find unescaped values

Think like an attacker

- According to your grails version
- Find unescaped values
- Use fuzzers

Think like an attacker

- According to your grails version
- Find unescaped values
- Use fuzzers
- Read and understand Samy code

Think like an attacker

- According to your grails version
- Find unescaped values
- Use fuzzers
- Read and understand Samy code
- Review OWASP XSS cheatsheets

Be aware

- Review your Grails app to double-check how all dynamic content gets escaped

Be aware

- Review your Grails app to double-check how all dynamic content gets escaped
- Monitor for suspicious traffic

Be aware

- Review your Grails app to double-check how all dynamic content gets escaped
- Monitor for suspicious traffic
- Spread the knowledge

Be aware

- Review your Grails app to double-check how all dynamic content gets escaped
- Monitor for suspicious traffic
- Spread the knowledge
- Adopt ZAP or similar fuzzers in your CI process

Be aware

- Review your Grails app to double-check how all dynamic content gets escaped
- Monitor for suspicious traffic
- Spread the knowledge
- Adopt ZAP or similar fuzzers in your CI process
- Review available security plugins for Grails

Application firewalls

- Enable common, safe rules

Application firewalls

- Enable common, safe rules
- Log unexpected traffic

Application firewalls

- Enable common, safe rules
- Log unexpected traffic
- Don’t fool yourself

Early-adopt CSP

- CSP: Content Security Policy

Early-adopt CSP

- CSP: Content Security Policy
- Adds headers to disable default behavior

Early-adopt CSP

- CSP: Content Security Policy
- Adds headers to disable default behavior
  - inline Javascript

Early-adopt CSP

- CSP: Content Security Policy
- Adds headers to disable default behavior
  - inline Javascript
  - dynamic code evaluation

Early-adopt CSP

- CSP: Content Security Policy
- Adds headers to disable default behavior
  - inline Javascript
  - dynamic code evaluation
- Still a Candidate Recommendation of W3C

Conclusions: Grails can defeat XSS

Grails

- Is able to defend our application from XSS attacks

Grails

- Is able to defend our application from XSS attacks
- But we need to pay attention to the details

Grails

- Is able to defend our application from XSS attacks
- But we need to pay attention to the details
- Upgrade to 2.3 ASAP

Grails

- Is able to defend our application from XSS attacks
- But we need to pay attention to the details
- Upgrade to 2.3 ASAP
- Pay attention to XSS

XSS

- Is much more dangerous than defacement jokes

XSS

- Is much more dangerous than defacement jokes
- The browsers are the actual target

XSS

- Is much more dangerous than defacement jokes
- The browsers are the actual target
- Difficult to monitor

XSS

- Is much more dangerous than defacement jokes
- The browsers are the actual target
- Difficult to monitor
- Unconfortable counter-measures in the browser: NoScript, Request Policy

Wake up

- Write secure applications by default

Wake up

- Write secure applications by default
- Get yourself used with Metasploit, Burp, ZAP

Wake up

- Write secure applications by default
- Get yourself used with Metasploit, Burp, ZAP
- Spread the word both horizontally and vertically

Picture credits

- **Cover**:
  `http://www.flickr.com/photos/usairforce/`
  CC by-nc
- **White rabbit**:
  `http://www.flickr.com/photos/alles-banane/5849593440`
  CC by-sa-nc
- **Hieroglyphs**:
  `http://www.flickr.com/photos/59372146@N00`
  CC by-sa-nc
- **Zombies**:
  `http://www.flickr.com/photos/aeviin/4986897433`
  CC by-sa-nc

XSS Countermeasures in Grails

Rafael Luque @rafael_luque — **OSOCO**
José San Leandro @rydnr — **Ventura24**

Groovy Madrid
Groovy Users Group