

Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks

Matthew Van Gundy and Hao Chen
University of California, Davis

16th Annual Network & Distributed System Security Symposium

Cross-Site Scripting (XSS) Vulnerabilities

Seclog: UCD Seclab Blog - archive.org for longevity - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.cs.ucdavis.edu/seclog/?p=24

Google

Seclog: UCD Seclab Blog

The UC Davis Security Lab's Blog

About Research Visitor Info IRB Guide

Home > This and that > archive.org for longevity

archive.org for longevity

September 28th, 2007 by Yuan

[Goto comments](#) [Leave a comment](#)

I've recently noticed that some of my comment spam contains author links in the form of `http://^a.archive.org/0-9/4/spammysite`. There are ~290 occurrences based on a quick sql query:

Whether archive.org, a very reputable highly ranked site, is crawled or not is not an issue. It doesn't even matter if they use "rel=nofollow". They are being exploited as enablers for spammers, who use archive.org's caching service to extend the otherwise short lifetime of their spammy site. Rather brilliant act on the part of the spammer.

In other news, e-mail spam could be a lot more than annoying...

[Spam](#), [This and that](#), [Vulnerabilities](#), [Web](#)

[Leave a comment](#) [Trackback](#)

Trackbacks (0) Comments (2)

No trackbacks yet.

Logged in as [MattVanGundy](#). [Logout](#)

[Subscribe to comments feed](#) [Submit Comment](#)

RSS feed

Recent Posts

- 1,474 Megapixel Inauguration Panorama
- BitArmor's No-Breach Guarantee
- Top 500 Worst Passwords
- If programming languages were religions...
- Soliciting readers: security blogs

Archives

February 2009	Archives
S M T W T F S	January 2009
1 2 3 4 5 6 7	December 2008
8 9 10 11 12 13 14	August 2008
15 16 17 18 19 20 21	February 2008
22 23 24 25 26 27 28	January 2008
= Jan	October 2007
	September 2007
	July 2007
	May 2007
	April 2007
	March 2007

Tags

[amusing](#) [Biometrics](#) [Blogs](#) [Code](#)

Done

Cross-Site Scripting (XSS) Vulnerabilities

Seclog: UCD Seclab Blog
The UC Davis Security Lab's Blog

Home > This and that > archive.org for longevity

archive.org for longevity

September 28th, 2007 by Yuan

I've recently noticed that some of my comment spam contains author links in the form of `http://*.archive.org[0-9]*/spammysite`. There are ~290 occurrences based on a quick sql query:

Whether `archive.org` is a semi-reputable link or not is not an issue. It is only used by spammers, who use `archive.org`'s caching service to extend the otherwise short lifetime of their spammy site. Rather brilliant act on the part of the spammer.

Comment: `<p class='comment'>{ $comment }</p>`

Leave a comment

Trackbacks (0) Comments (2)

No trackbacks yet.

Logged in as **MattVanGundy**. Logout =>

Subscribe to comments feed

Submit Comment

Done

RSS feed

Recent Posts

- 1,474 Megapixel Inauguration Panorama
- BRArmor's No-Breach Guarantee
- Top 500 Worst Passwords
- If programming languages were religions...
- Soliciting readers: security blogs

Archives

- January 2009
- December 2008
- August 2008
- February 2008
- January 2008
- October 2007
- September 2007
- July 2007
- May 2007
- April 2007
- March 2007

Tags

amusing Biometrics Blogs Code

Cross-Site Scripting (XSS) Vulnerabilities

Seclog: UCD Seclab Blog
The UC Davis Security Lab's Blog

Home > This and that > archive.org for longevity

archive.org for longevity

September 28th, 2007 by Yuan

I've recently noticed that some of my comment spam contains author links in the form of `http://*.archive.org/0-9/*spammysite`. There are ~290 occurrences based on a quick sql query.

Whether archive.org is a semi-reputable hub/warehouse site, or not is not an issue. It is a good idea to use it for my use, but being exploited as enablers for spammers, who use archive.org's caching service to extend the otherwise short lifetime of their spammy site. Rather brilliant act on the part of the spammer.

Great Article!

Spam, This and that, Vulnerabilities, Web

Leave a comment Trackback

No trackbacks yet.

Logged in as **MattVanGundy**. Logout »

Subscribe to comments feed **Submit Comment**

Done

RSS feed

Recent Posts

- 1,474 Megapixel Inauguration Panorama
- BRArmor's No-Breach Guarantee
- Top 500 Worst Passwords
- If programming languages were religions...
- Soliciting readers: security blogs

Archives

February 2009	January 2009
S M T W T F S	December 2008
1 2 3 4 5 6 7	August 2008
8 9 10 11 12 13 14	February 2008
15 16 17 18 19 20 21	January 2008
22 23 24 25 26 27 28	October 2007
« Jan	September 2007
	July 2007
	May 2007
	April 2007
	March 2007

Tags

amusing Biometrics Blogs Code

Cross-Site Scripting (XSS) Vulnerabilities

Seclog: UCD Seclab Blog
The UC Davis Security Lab's Blog

Home > This and that > archive.org for longevity

archive.org for longevity

September 28th, 2007 by Yuan

I've recently noticed that some of my comment spam contains author links in the form of `http://*.archive.org[0-9]*/spammysite`. There are ~290 occurrences based on a quick sql query:

When `archive.org` is used as a spammer's author link, it is not an issue. It is only used as an enabler for spammers, who use `archive.org`'s caching service to extend the otherwise short lifetime of their spammy site. Rather brilliant act on the part of the spammer.

Spam, This and that, Vulnerabilities, Web

Leave a comment Trackback

No trackbacks yet.

Logged in as **MattVanGundy**. Logout »

Subscribe to comments feed Submit Comment

Done

RSS feed

Recent Posts

- 1,474 Megapixel Inauguration Panorama
- BRArmor's No-Breach Guarantee
- Top 500 Worst Passwords
- If programming languages were religions...
- Soliciting readers: security blogs

Archives

- January 2009
- December 2008
- August 2008
- February 2008
- January 2008
- October 2007
- September 2007
- July 2007
- May 2007
- April 2007
- March 2007

Tags

amusing Biometrics Blogs Code

Cross-Site Scripting (XSS) Vulnerabilities

Seclog: UCD Seclab Blog
The UC Davis Security Lab's Blog

Home > This and that > archive.org for longevity

archive.org for longevity

September 28th, 2007 by Yuan

I've recently noticed that some of my comment spam contains author links in the form of `http://*.archive.org/0-9/*spammysite`. There are ~290 occurrences based on a quick sql query:

Whether archive.org, a semi-reputable but heavily used site, is exploited or not is not an issue. It is the fact that my users are being exploited as enablers for spammers, who use archive.org's caching service to extend the otherwise short lifetime of their spammy site. Rather brilliant act on the part of the spammer.

Spam, This and that, Vulnerabilities, Web

Leave a comment

No trackbacks yet.

Logged in as **MattVanGundy**. Logout =>

Subscribe to comments feed

Submit Comment

Done

RSS feed

Recent Posts

- 1,474 Megapixel Inauguration Panorama
- BRArmor's No-Breach Guarantee
- Top 500 Worst Passwords
- If programming languages were religions...
- Solving readers: security blogs

Archives

- January 2009
- December 2008
- August 2008
- February 2008
- January 2008
- October 2007
- September 2007
- July 2007
- May 2007
- April 2007
- March 2007

Tags

amusing Biometrics Blogs Code

<p class='comment'>
</p><script>p0wn () </script><p>
</p>

Threat Model

- ▶ An attacker can submit arbitrary content to XSS-vulnerable applications
- ▶ An attacker cannot compromise web server or browser directly
- ▶ Malicious content must contain XHTML tags and attributes

Limitations of Existing Solutions

Server-side

- ▶ Server *sanitizes* untrusted data before sending it to the client
- ▶ Client may interpret data in an unexpected way
- ▶ E.g. Server replaces "<script>" with ""
But attacker injects `<script/xss>`

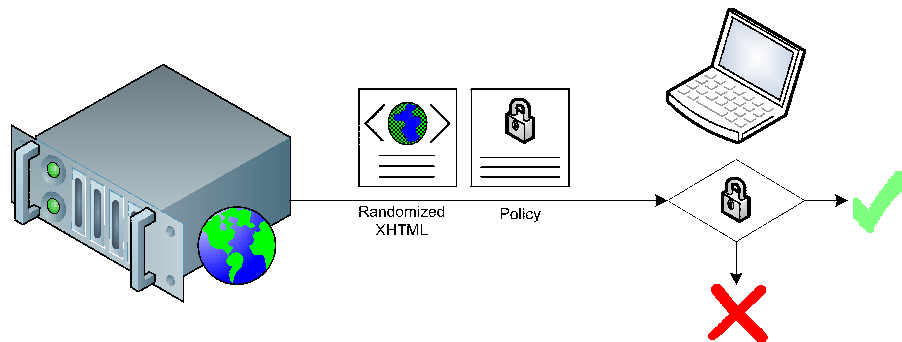
Client-side

- ▶ Client enforces a server-specified policy

Challenges

- ▶ The client must know whether to trust content
- ▶ Attacker must not be able to forge trust metadata

Noncespaces Architecture



- ▶ Server partitions content into *trust classes*
- ▶ Server randomizes document to prevent forging of trust classification
- ▶ Server specifies policy of content permitted for each trust class
- ▶ Client displays the document only if it conforms to the policy

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("<http://www.w3.org/1999/xhtml>", "q")

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("<http://www.w3.org/1999/xhtml>", "q")
- ▶ FAQML question = ("urn:FAQML", "q")

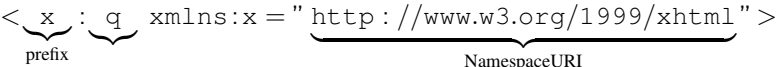
Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("http://www.w3.org/1999/xhtml", "q")
- ▶ FAQML question = ("urn:FAQML", "q")
- ▶ `< x : q xmlns:x = " http://www.w3.org/1999/xhtml " >`

Namespaces in XML

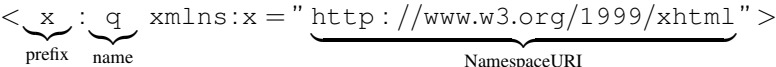
- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("http://www.w3.org/1999/xhtml", "q")
- ▶ FAQML question = ("urn:FAQML", "q")
- ▶ `< x : q xmlns:x = " http://www.w3.org/1999/xhtml " >`
NamespaceURI

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("http://www.w3.org/1999/xhtml", "q")
- ▶ FAQML question = ("urn:FAQML", "q")
- ▶ `<x : q xmlns:x = "http://www.w3.org/1999/xhtml" >`


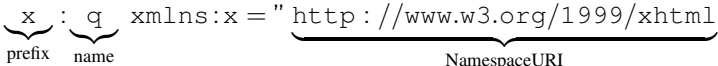
The diagram shows the XML declaration `<x : q xmlns:x = "http://www.w3.org/1999/xhtml" >` with annotations. A bracket under the `x` is labeled "prefix". A bracket under the `q` is labeled "local name". A large bracket under the entire `xmlns:x = "http://www.w3.org/1999/xhtml"` is labeled "NamespaceURI".

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("http://www.w3.org/1999/xhtml", "q")
- ▶ FAQML question = ("urn:FAQML", "q")
- ▶ `< x : q xmlns:x = " http://www.w3.org/1999/xhtml " >`


The diagram shows the XML namespace declaration `<x:q xmlns:x="http://www.w3.org/1999/xhtml">` with three annotations: a bracket under 'x' labeled 'prefix', a bracket under 'q' labeled 'name', and a bracket under the entire URI string labeled 'NamespaceURI'.

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("http://www.w3.org/1999/xhtml", "q")
- ▶ FAQML question = ("urn:FAQML", "q")
- ▶ `< x : q xmlns:x = " http://www.w3.org/1999/xhtml " >`

- ▶ `<f:q xmlns:f="urn:FAQML">`

Namespaces in XML

- ▶ In (X)HTML: `<q>` = quote, `<a>` = anchor
- ▶ In FAQML: `<q>` = question, `<a>` = answer
- ▶ XHTML quote = ("http://www.w3.org/1999/xhtml", "q")
- ▶ FAQML question = ("urn:FAQML", "q")
- ▶ `< x : q xmlns:x = " http://www.w3.org/1999/xhtml " >`
prefix name NamespaceURI
- ▶ `<f:q xmlns:f="urn:FAQML">`
- ▶ `<faq:q xmlns:faq="urn:FAQML">`

Defeating Node Splitting

▶ `<x:a>...</x:a>`

Defeating Node Splitting

▶ `<x:a>...</x:a>`

▶ `<x:a>...`

Defeating Node Splitting

- ▶ `<x:a>...</x:a>`
- ▶ `<x:a>...`
- ▶ `<x:a>...</y:a>`

Encoding Trust Classifications

- ▶ **Trusted** <a>

Encoding Trust Classifications

- ▶ **Trusted** $\langle a \rangle \Rightarrow \langle t : a \rangle$

Encoding Trust Classifications

- ▶ **Trusted** $\langle a \rangle \Rightarrow \langle t : a \rangle$
- ▶ **Untrusted** $\langle a \rangle$

Encoding Trust Classifications

- ▶ **Trusted** $\langle a \rangle \Rightarrow \langle t : a \rangle$
- ▶ **Untrusted** $\langle a \rangle$
- ▶ Randomly choose trusted prefixes to prevent forgery

Web Page Before Noncespaces

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>nile.com : ++Shopping</title>
</head>
<body>
<h1 id="title">{$item->name}</h1>

<h2>Reviews</h2>
<p class='review'>
  {$review}
</p>
</body>
</html>
```

Node Splitting Attack After Noncespaces

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<r617:html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:r617="http://www.w3.org/1999/xhtml">
<r617:head>
    <r617:title>nile.com : ++Shopping</r617:title>
</r617:head>
<r617:body>
<r617:h1 r617:id="title">Useless Do-dad</r617:h1>

<r617:h2>Reviews</r617:h2>
<r617:p r617:class='review'>
    </p>  <script>p0wn()</script>  <p>
</r617:p>
</r617:body>
</r617:html>
```

XSS Attack After Noncespaces

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<r617:html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:r617="http://www.w3.org/1999/xhtml">
<r617:head>
    <r617:title>nile.com : ++Shopping</r617:title>
</r617:head>
<r617:body>
<r617:h1 r617:id="title">Useless Do-dad</r617:h1>

<r617:h2>Reviews</r617:h2>
<r617:p r617:class='review'>
    <script src='http://badguy.com/p0wn.js' />
</r617:p>
</r617:body>
</r617:html>
```

Need for a client-side policy

Innocuous Input

`WARNING:`

Need for a client-side policy

Innocuous Input

`WARNING:`

`very important`

Need for a client-side policy

Innocuous Input

`WARNING:`

`very important`

`[1]`

Need for a client-side policy

Innocuous Input

`WARNING:`

`very important`

`[1]`

Malicious Input

`<b onmouseover='...'>WARNING:`

Need for a client-side policy

Innocuous Input

```
<b>WARNING:</b>
```

```
<em>very</em> important
```

```
<a href='http://useful.com/'>[1]</a>
```

Malicious Input

```
<b onmouseover='...'>WARNING:</b>
```

```
<em onclick='...'>very</em> important
```

Need for a client-side policy

Innocuous Input

```
<b>WARNING:</b>
```

```
<em>very</em> important
```

```
<a href='http://useful.com/'>[1]</a>
```

Malicious Input

```
<b onmouseover='...'>WARNING:</b>
```

```
<em onclick='...'>very</em> important
```

```
<a href=' javascript:...''>[1]</a>
```

Need for a client-side policy

XHTML

Policy

``

``

``

Need for a client-side policy

XHTML

``

``

``

Policy

`allow //untrusted:b`

Need for a client-side policy

XHTML

``

``

``

Policy

`allow //untrusted:b`

`allow //untrusted:em`

Need for a client-side policy

XHTML

``

``

``

Policy

```
allow //untrusted:b
```

```
allow //untrusted:em
```

```
allow //untrusted:a/@untrusted:href[
  starts-with(normalize-space(.),
              "http:")]
```

Need for a client-side policy

XHTML

``

``

``

`<b onmouseover=' '>`

`<em onclick=' '>`

``

Policy

`allow //untrusted:b`

`allow //untrusted:em`

`allow //untrusted:a/@untrusted:href[
 starts-with(normalize-space(.),
 "http:")]`

Need for a client-side policy

XHTML

``

``

``

`<b onmouseover=' '>`

`<em onclick=' '>`

``

Policy

`allow //untrusted:b`

`allow //untrusted:em`

`allow //untrusted:a/@untrusted:href[
 starts-with(normalize-space(.),
 "http:")]`

`deny //@untrusted:onmouseover`

Need for a client-side policy

XHTML

``

``

``

`<b onmouseover=' '>`

`<em onclick=' '>`

``

Policy

`allow //untrusted:b`

`allow //untrusted:em`

`allow //untrusted:a/@untrusted:href[
 starts-with(normalize-space(.),
 "http:")]`

`deny //@untrusted:onmouseover`

`deny //@untrusted:*`

Need for a client-side policy

XHTML

``

``

``

`<b onmouseover=' '>`

`<em onclick=' '>`

``

Policy

`allow //untrusted:b`

`allow //untrusted:em`

`allow //untrusted:a/@untrusted:href[
 starts-with(normalize-space(.),
 "http:")]`

`deny //@untrusted:onmouseover`

`deny //@untrusted:*`

`deny //@untrusted:href[
 starts-with(normalize-space(.),
 "javascript:")]`

Determining Trusted Content

- ▶ Design patterns separate presentation and business logic
- ▶ Templates contain static HTML (presentation)
- ▶ Program creates dynamic content from user input

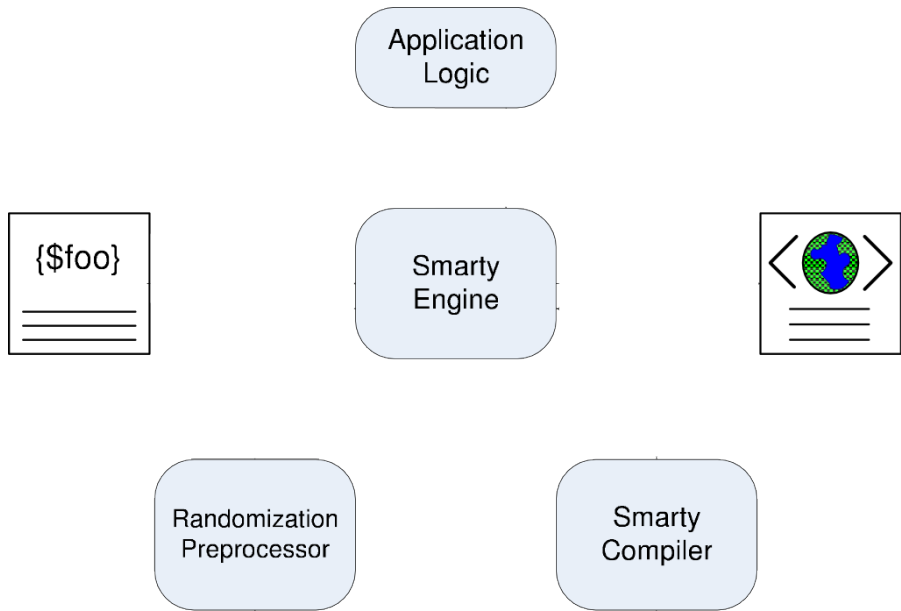
Determining Trusted Content

- ▶ Design patterns separate presentation and business logic
- ▶ Templates contain static HTML (presentation)
 - ▶ Classify as trusted
- ▶ Program creates dynamic content from user input

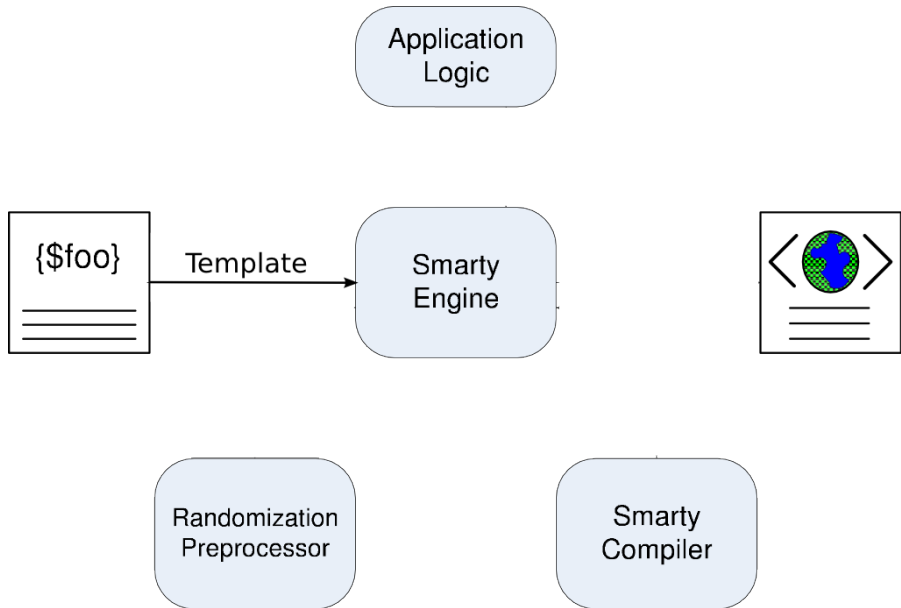
Determining Trusted Content

- ▶ Design patterns separate presentation and business logic
- ▶ Templates contain static HTML (presentation)
 - ▶ Classify as trusted
- ▶ Program creates dynamic content from user input
 - ▶ Classify as untrusted

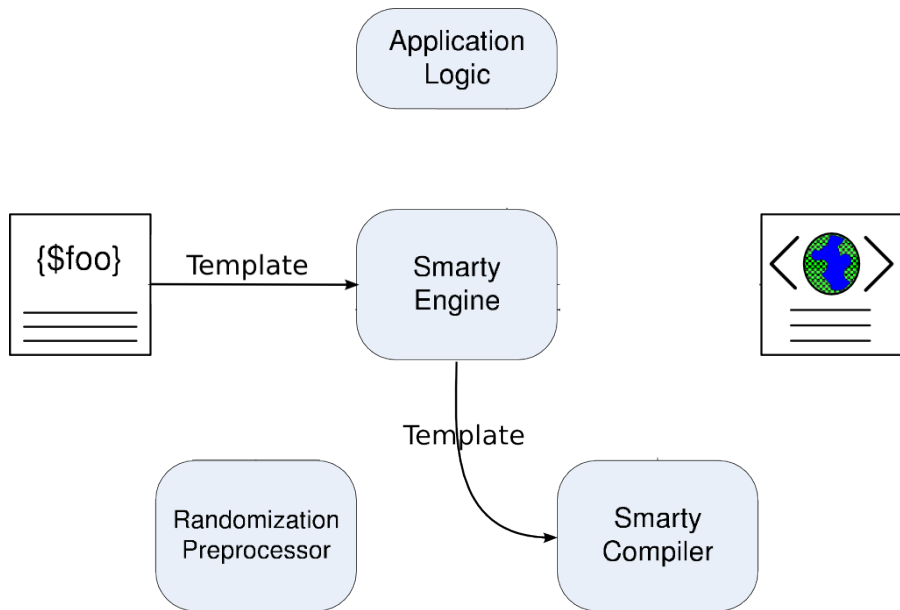
Modifications to Smarty



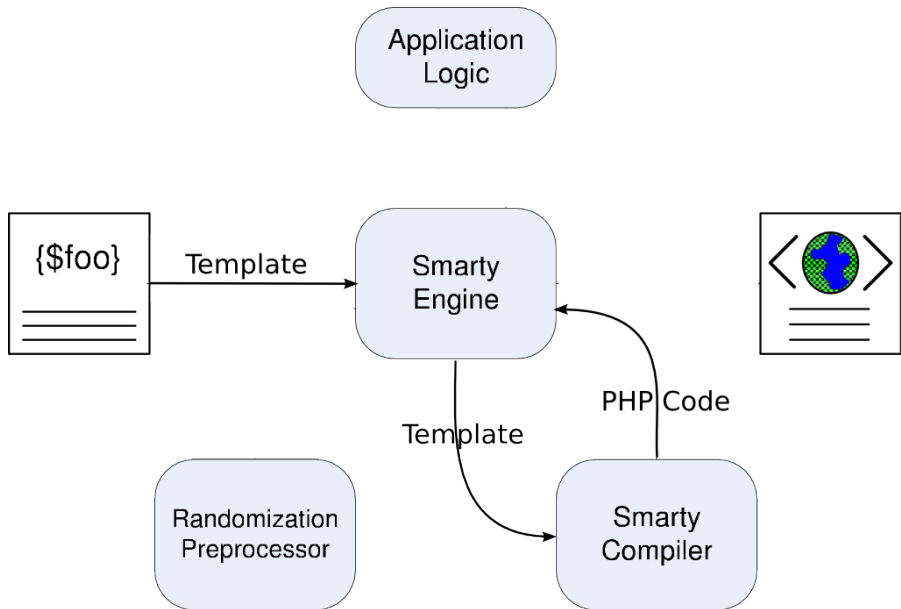
Modifications to Smarty



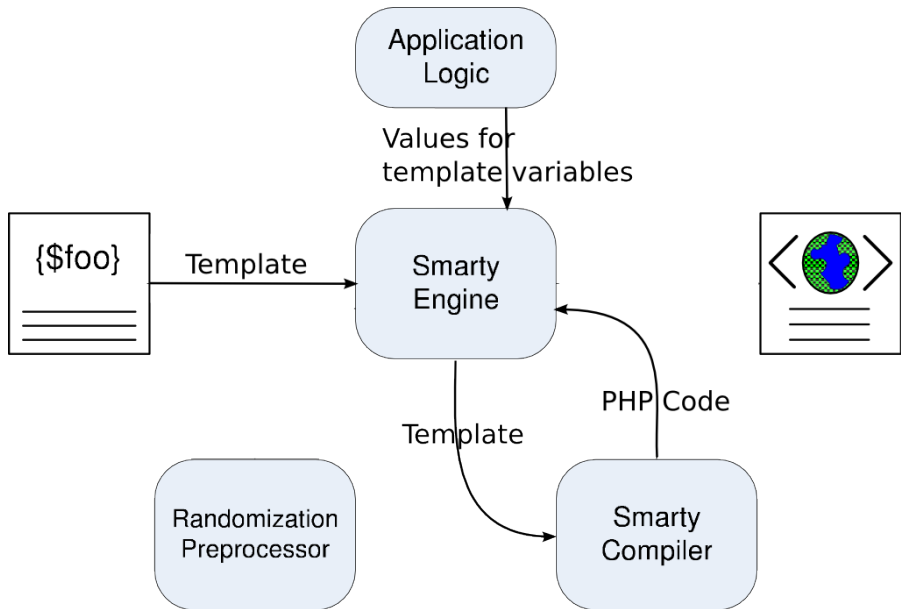
Modifications to Smarty



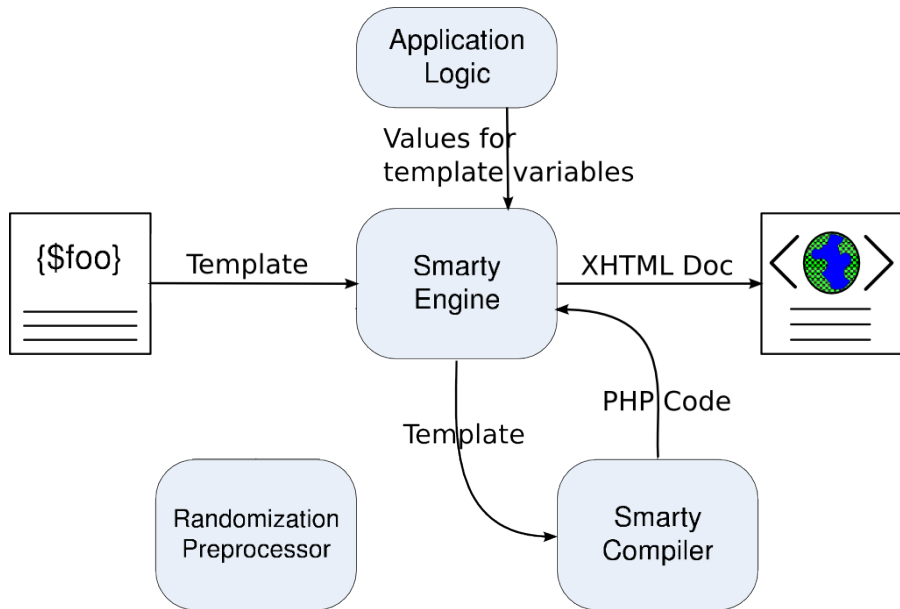
Modifications to Smarty



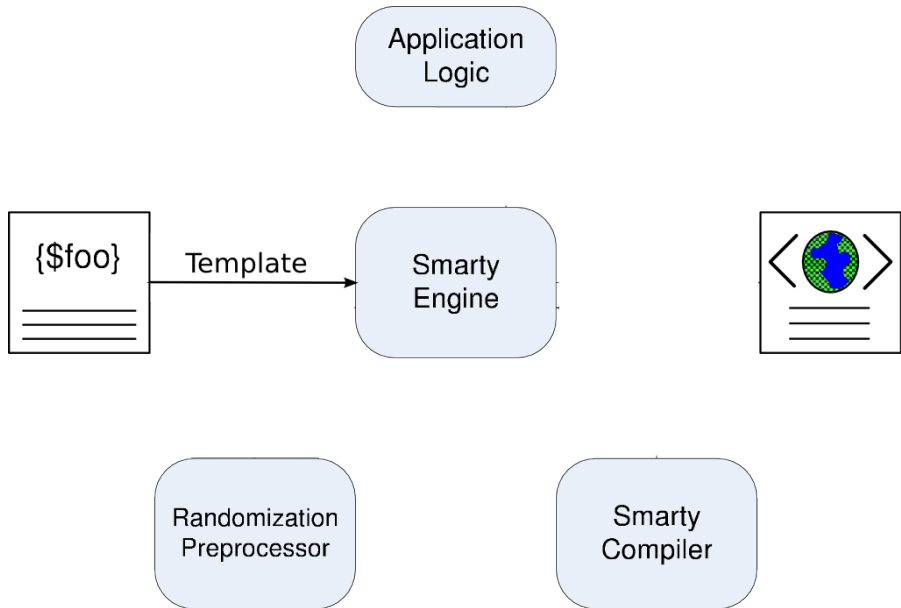
Modifications to Smarty



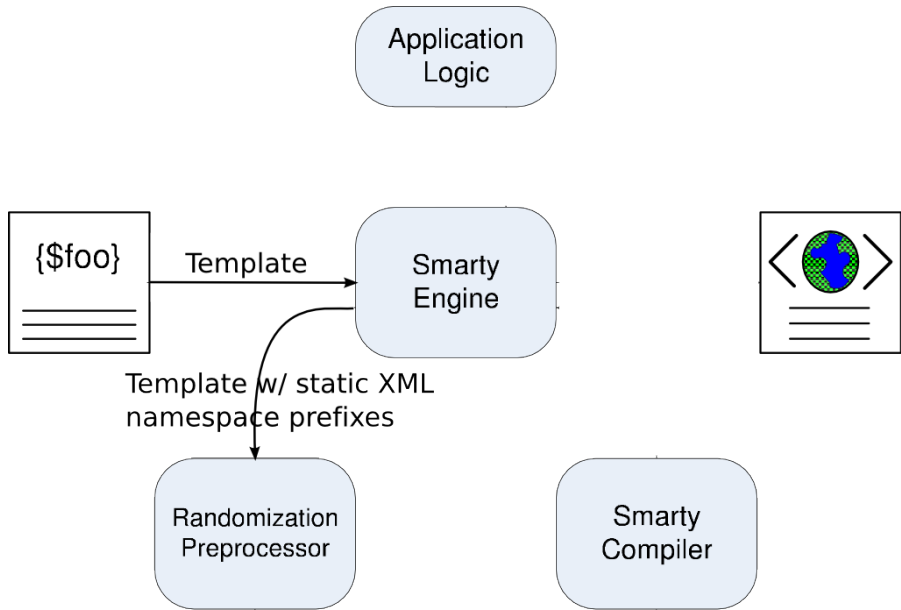
Modifications to Smarty



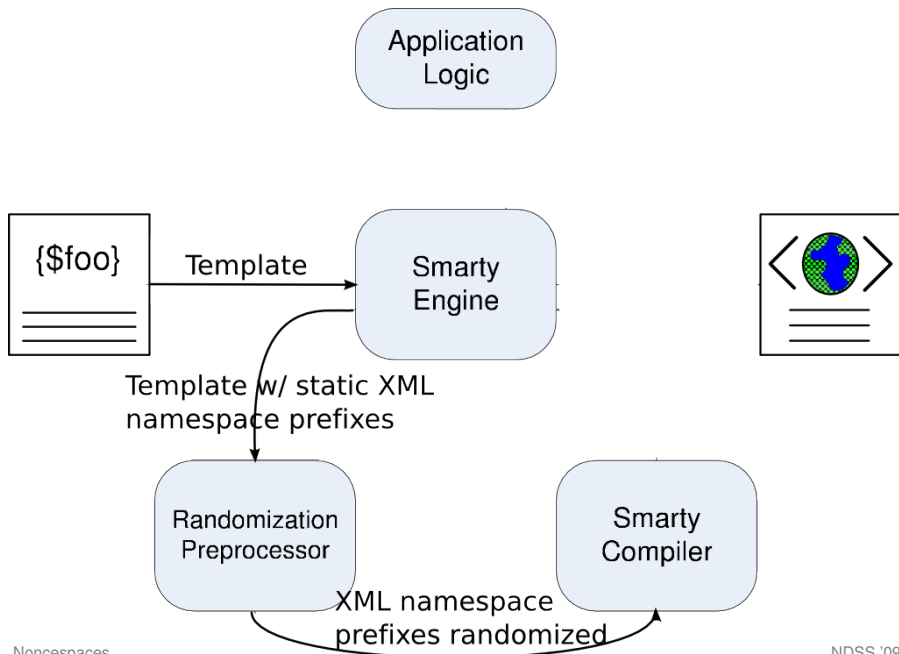
Modifications to Smarty



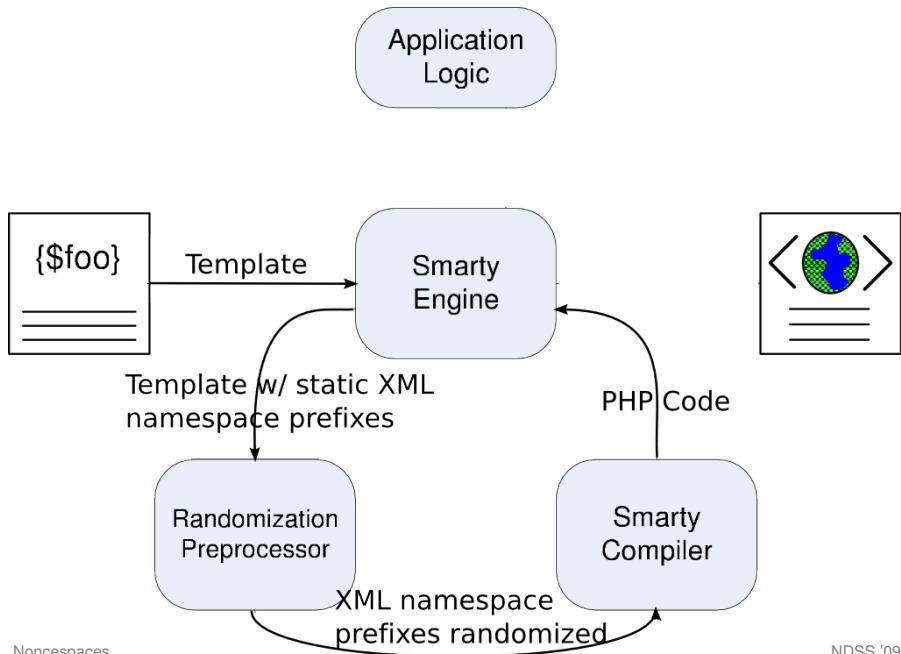
Modifications to Smarty



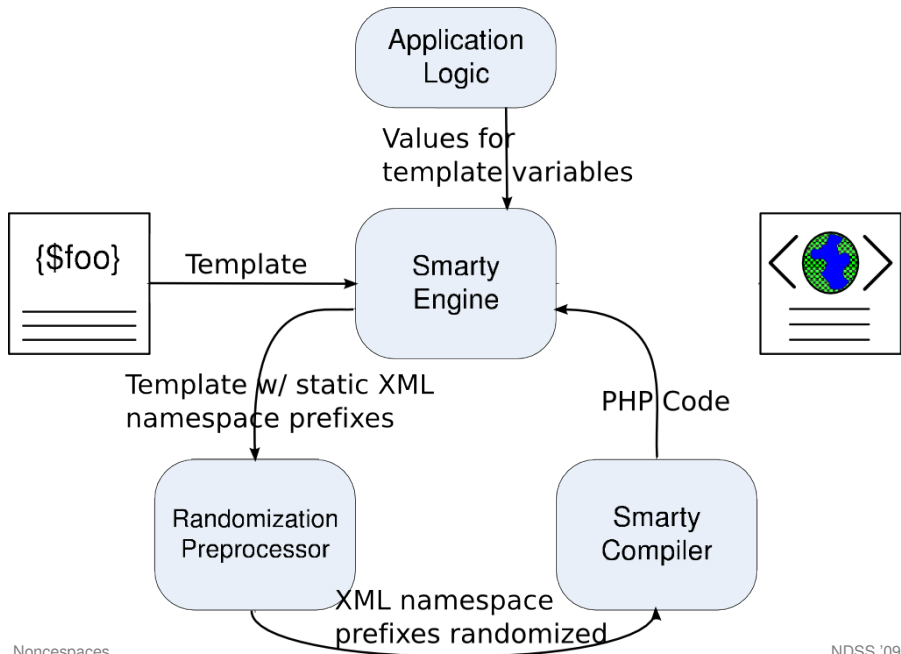
Modifications to Smarty



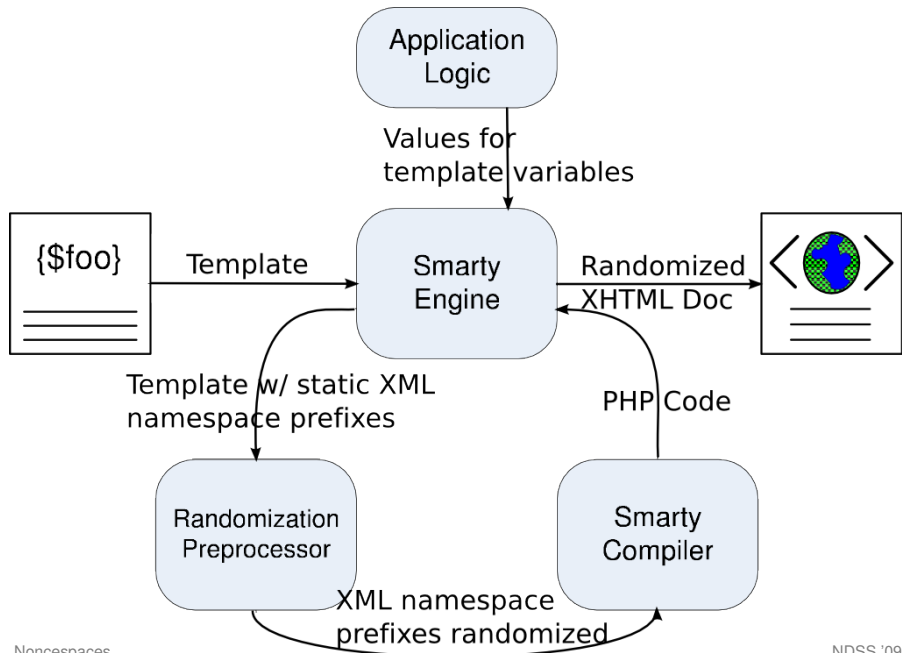
Modifications to Smarty



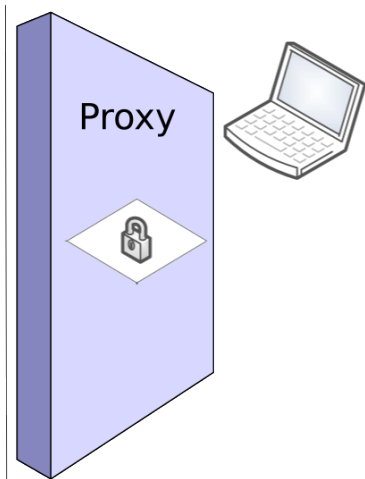
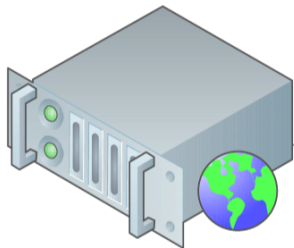
Modifications to Smarty



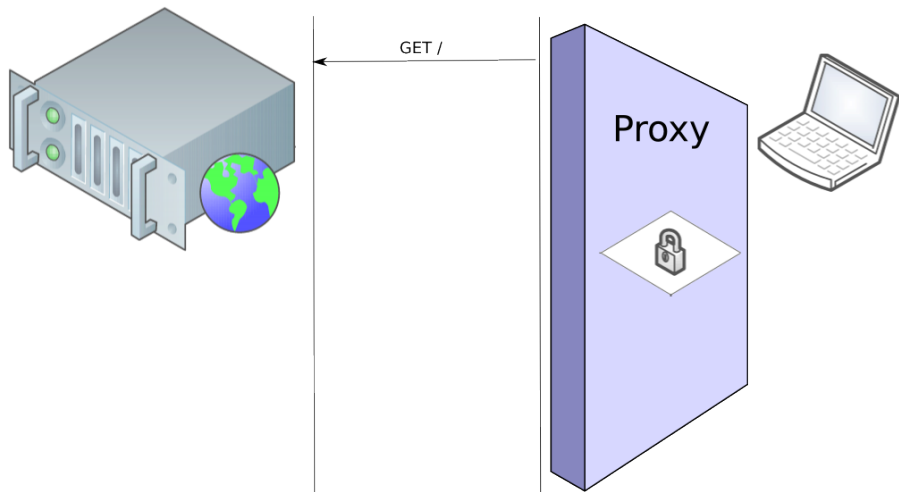
Modifications to Smarty



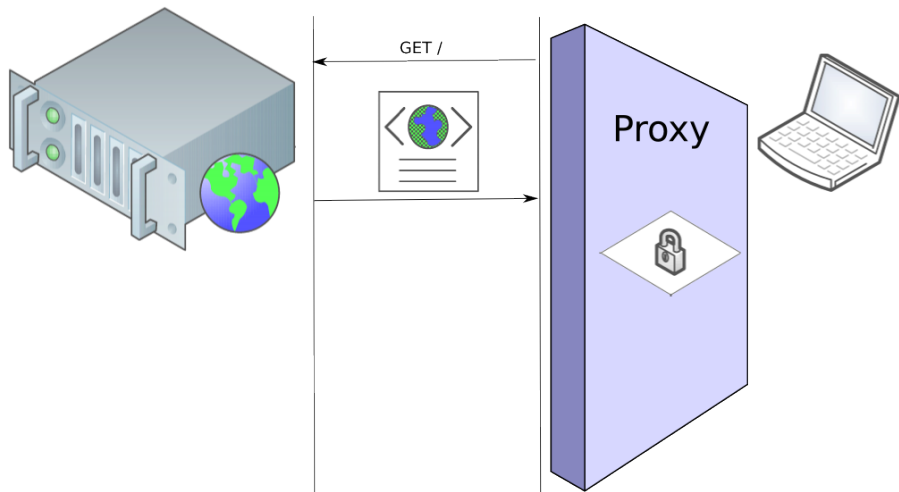
Client-side Modifications



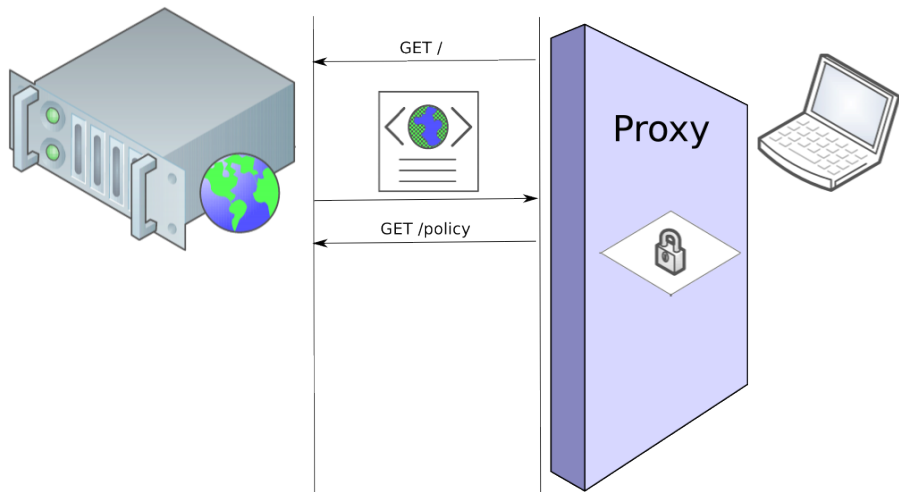
Client-side Modifications



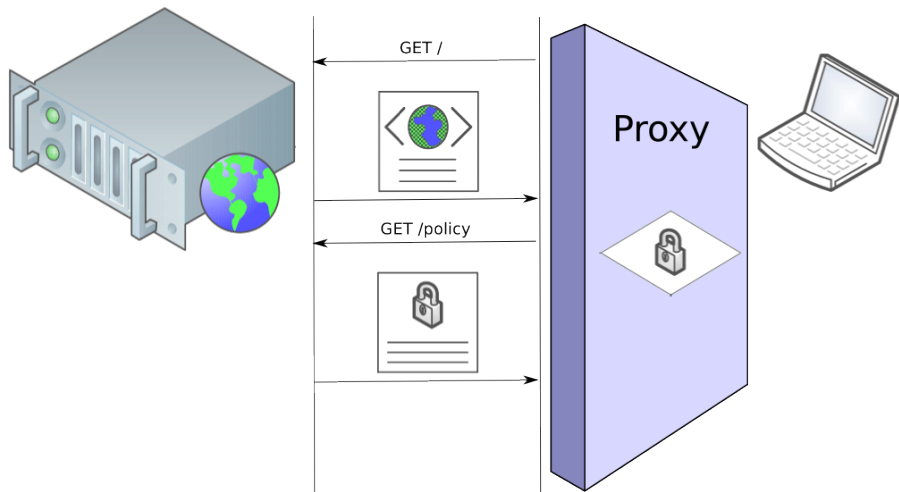
Client-side Modifications



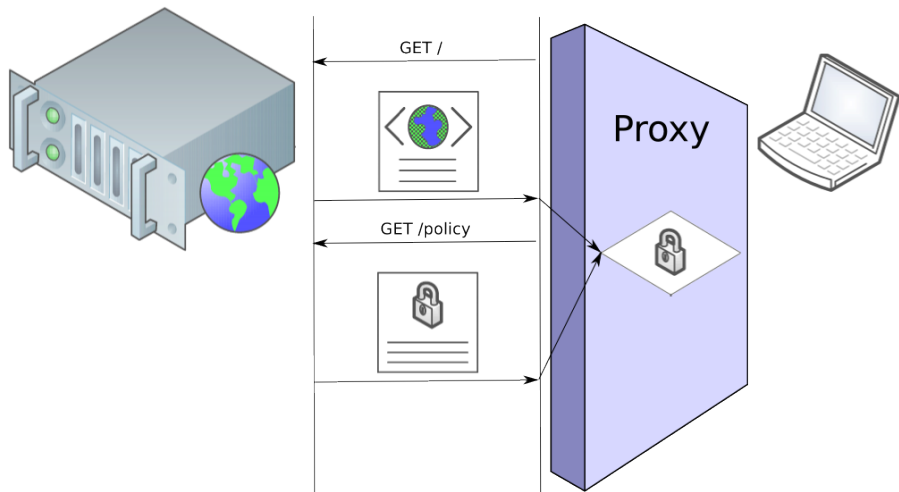
Client-side Modifications



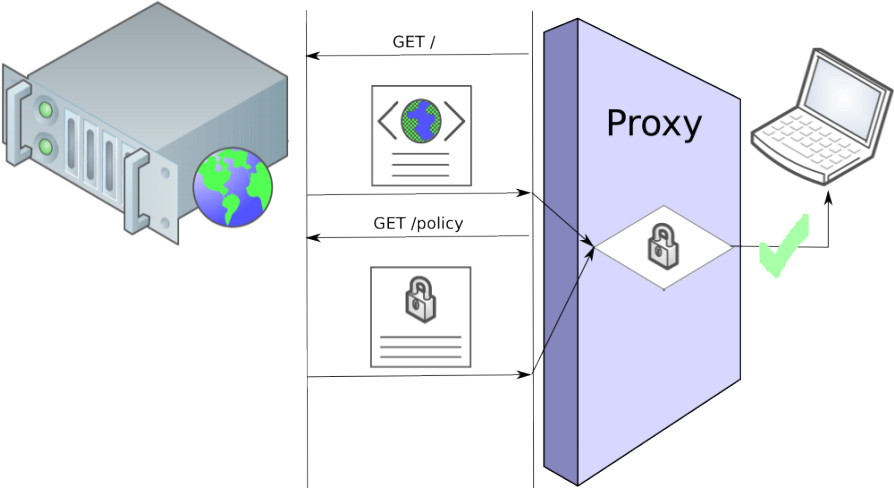
Client-side Modifications



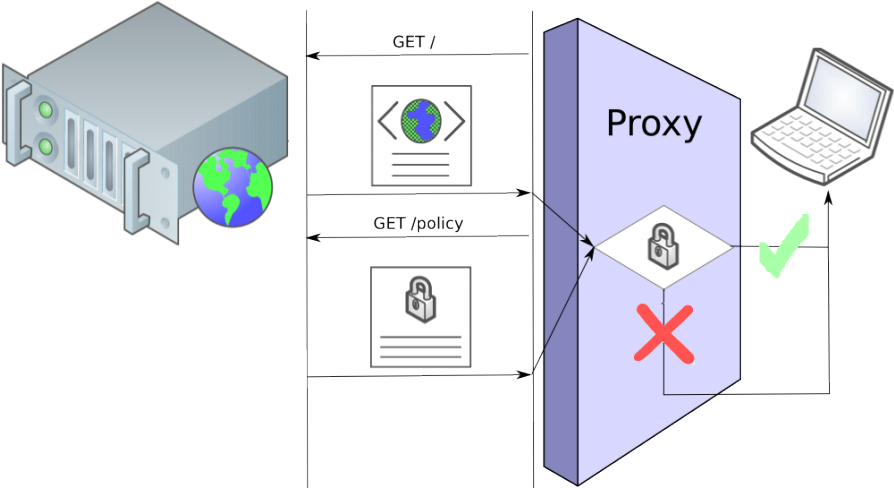
Client-side Modifications



Client-side Modifications



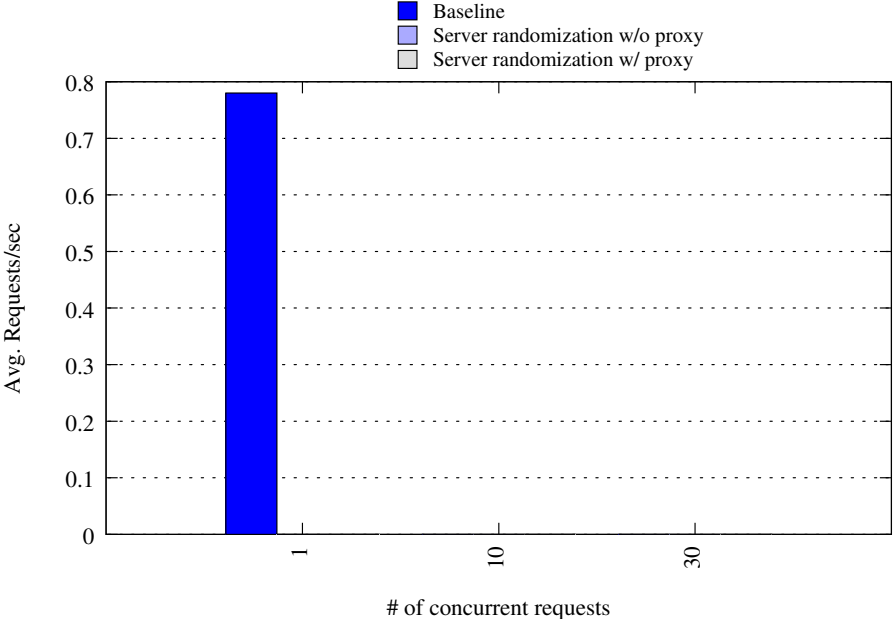
Client-side Modifications



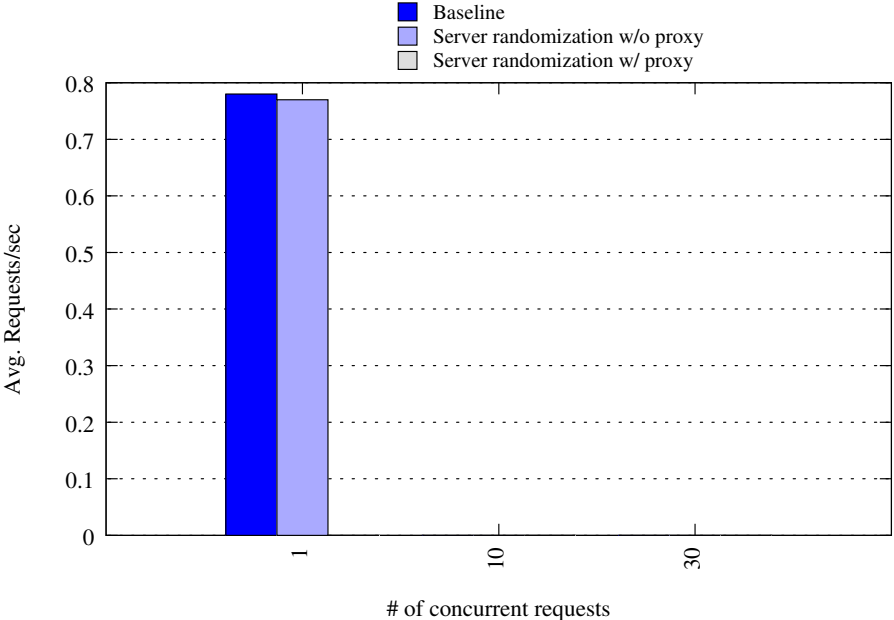
Evaluation

- ▶ Tested effectiveness of Noncespaces on 2 applications
- ▶ Developed policy for each application
- ▶ Ensured that Noncespaces stopped a number of XSS attacks
- ▶ Measured performance overhead of both server-side randomization and client-side policy checking

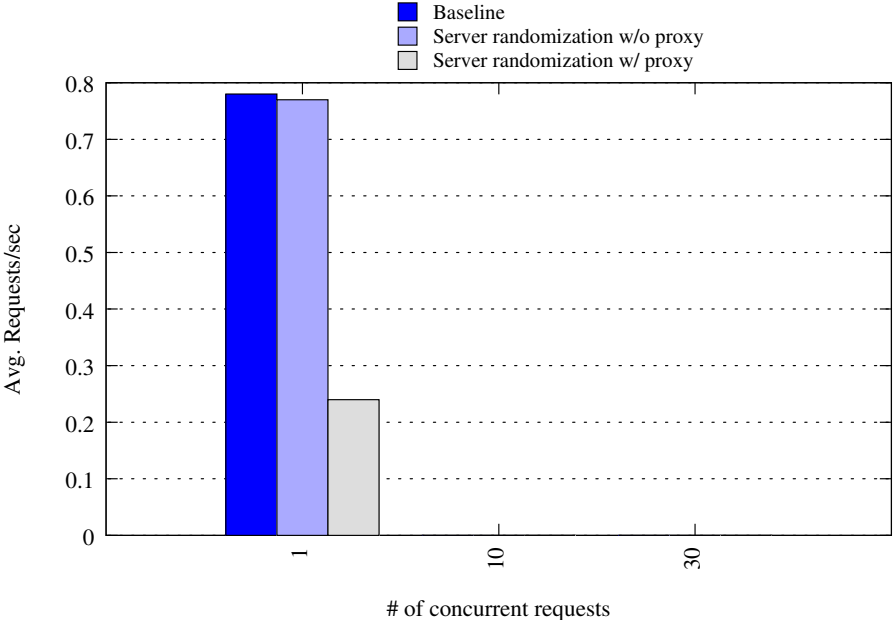
Evaluation



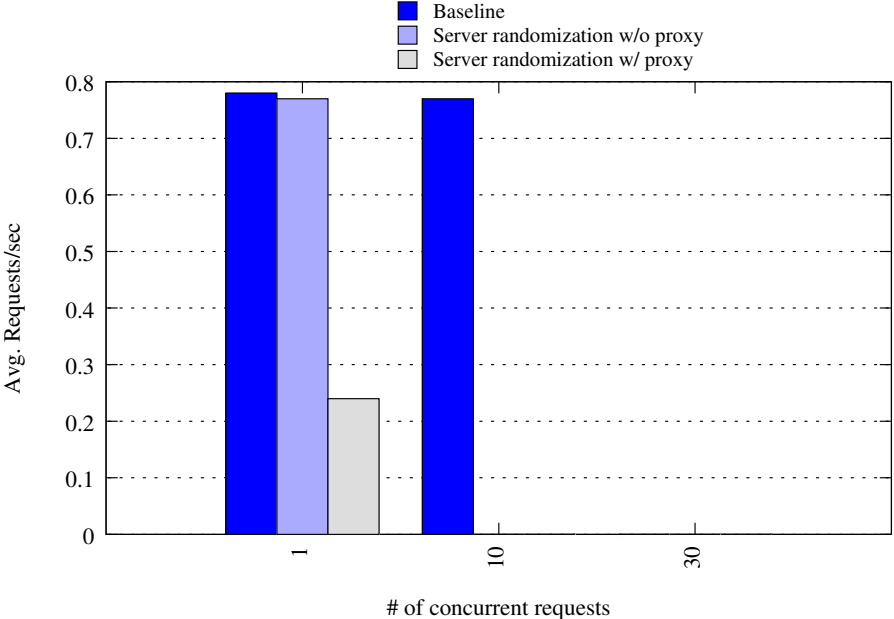
Evaluation



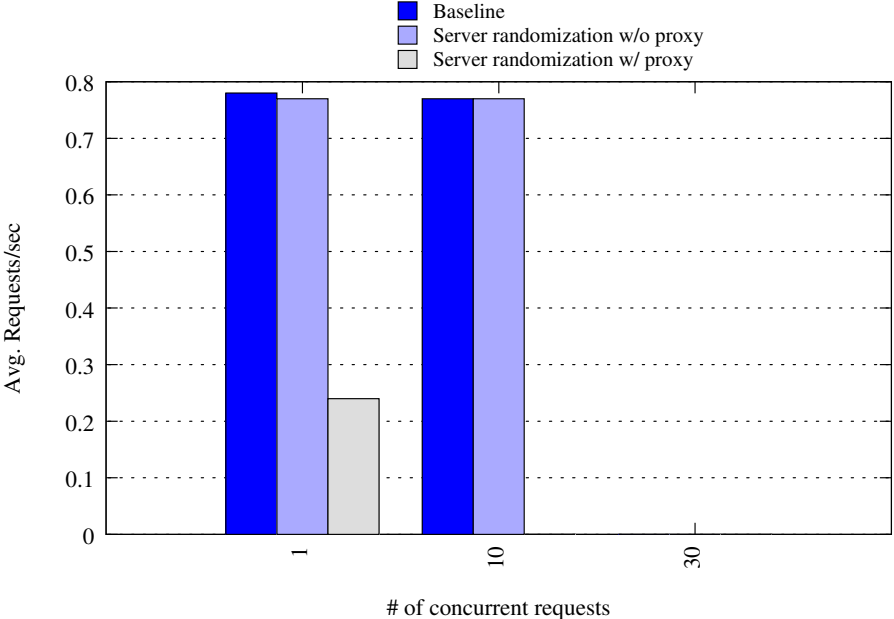
Evaluation



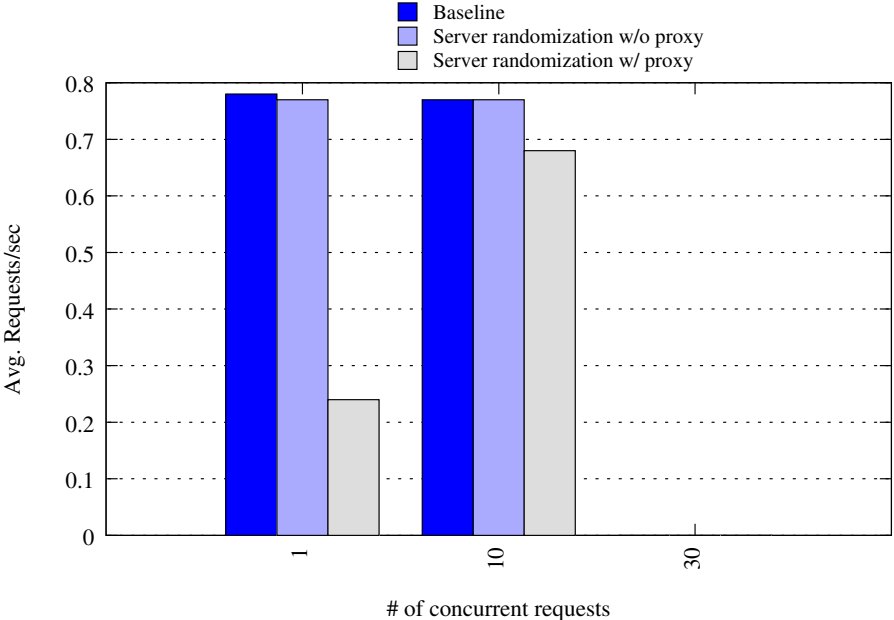
Evaluation



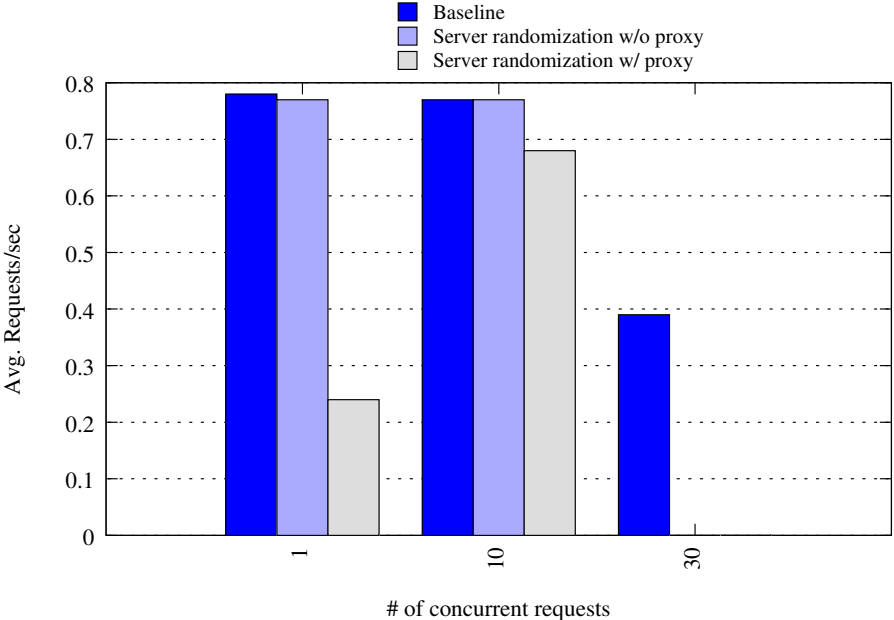
Evaluation



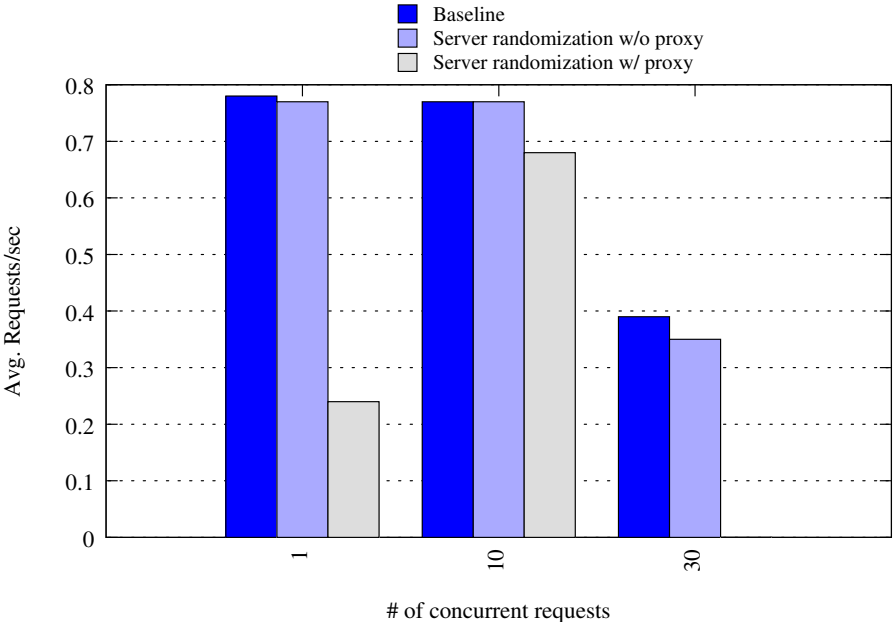
Evaluation



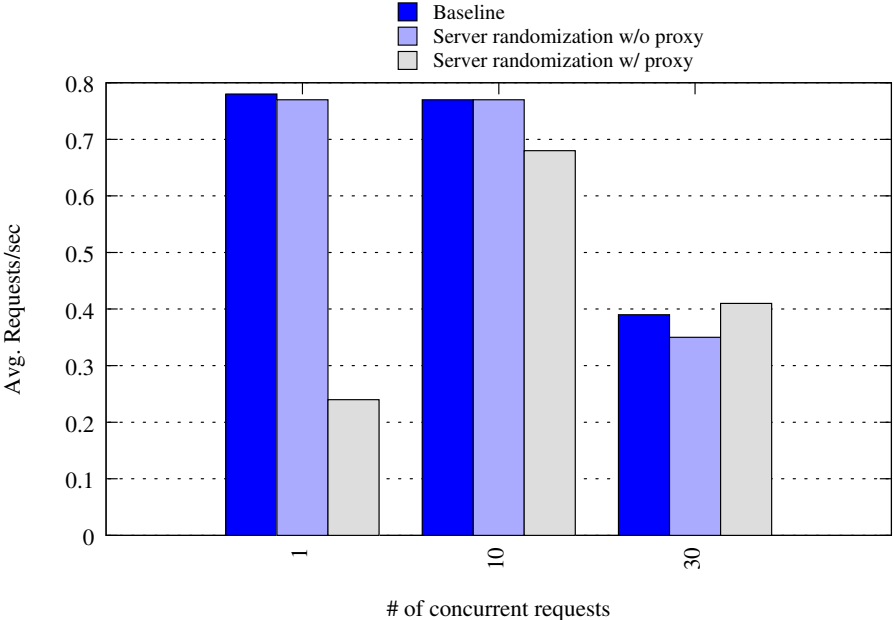
Evaluation



Evaluation



Evaluation



Related Work

- ▶ Instruction Set Randomization (Kc et al., CCS '03) and (Barrantes et al., CCS '03)
- ▶ BEEP (Jim et al., WWW '07)
- ▶ Mutation Event Transforms (Erlingsson et al., HotOS '07)
- ▶ Noxes (Kirda et al., ACM SAC '06)
- ▶ Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis (Vogt et al., NDSS '07)

Conclusion

- ▶ We can achieve security without data sanitization on the server
 - ▶ Servers classify how trustworthy content is
 - ▶ Servers convey trust classifications in a tamper resistant way
 - ▶ Clients interpreting the content enforce the policy
- ▶ Leverage randomization and XML features to thwart XSS attacks
- ▶ Leverage design paradigms to determine trust information without dynamic information flow tracking

Questions?

Example Noncespaces Policy

```
1 namespace trusted
2 namespace untrusted
3
4 allow //trusted:*
5 allow //trusted:@*
6
7 allow //untrusted:b
8 allow //untrusted:i
9 allow //untrusted:u
10 allow //untrusted:a
11 allow //untrusted:a/@untrusted:href[
12     starts-with(normalize-space(.), "http:")]
13 allow //untrusted:img
14 allow //untrusted:img/@untrusted:src[
15     starts-with(normalize-space(.), "http:")]
16
17 deny //*
18 deny //@*
```